ELSEVIER

Theoretical Computer Science 206 (1998) 219–256

Theoretical Computer Science

A uniform approach to true-concurrency and interleaving semantics for Petri nets

Lutz Priese*, Harro Wimmel

Universität Koblenz-Landau, D-56075 Koblenz, Germany

Received December 1995
Communicated by M. Nivat

Abstract

This paper continues a research on universal contexts and semantics for Petri nets started by Nielsen, Priese and Sassone (1995). We consider generalized, labelled Petri nets N where some transitions and places are distinguished as *public*. They form the *interface* of N that may communicate with a Petri net *context*. An elementary calculus, $\mathscr{E}$, is introduced in which one can construct any Petri net with an interface from trivial constants (single places, single transitions) by drawing arcs, adding tokens, and hiding public places and transitions. We prove the existence of a *universal context* U such that two Petri nets behave the same in any context if their behaviour is equal in the universal context. Let $B(U[N])$ be the behaviour of N embedded in its universal context, where B may be the interleaving language, step-language, or true-concurrent pomset-language. In any of these cases, $B(U[N])$ (in contrast to $B(N)$) turns out to be a compositional semantics of N with respect to the algebra $\mathscr{E}$. In addition, the interleaving- and step-semantics are just special cases of the true-concurrency pomset-semantics.

Keywords: Petri nets; Compositional semantics; Interleaving and true-concurrency models

1. Introduction

This chapter presents an outline of the intentions of this paper. Formal definitions and proofs of the concepts and facts mentioned here are in the following chapters.

1.1. Some general remarks on semantics and Petri nets

On a very abstract level, a behaviour B for a programming language $\mathscr{PL}$ is simply a mapping $B: \mathscr{PL} \to \mathscr{M}$ that attaches to any program $p \in \mathscr{PL}$ a meaning $B(p)$ in $\mathscr{M}$. $\mathscr{M}$ is some class of reasonably well understood mathematical objects, e.g., sets of words or partial orders, domains, (ultra-)metric complete spaces, etc. $\mathscr{PL}$ usually

* Corresponding author. E-mail: priese@uni-koblenz.de.

0304-3975/98/$19.00
PII S0304-3975(97)00169-2

possesses some algebraic structure $(\mathscr{PL}, \{op_i^k\}_{i\in I, k\in K})$ of k-ary syntactic operators $op_i^k : \mathscr{PL}^k \to \mathscr{PL}$. In this paper, by a semantics $\mathscr{S}$ we understand a compositional behaviour. Compositionality of $\mathscr{S}$ (with respect to this algebra of $\mathscr{PL}$) simply means the existence of semantic operators $\widehat{op}_i^k : \mathscr{M}^k \to \mathscr{M}$ such that $\mathscr{S}(op_i^k(p_1,\ldots,p_k)) = \widehat{op}_i^k(\mathscr{S}(p_1),\ldots,\mathscr{S}(p_k))$ holds for all p_j in $\mathscr{PL}$. Some concept of a *context* is often used where $C[p]$ denotes the program p embedded into the context C. We define a context as an expression of the algebra $(\mathscr{PL}\cup\{\bullet\}, \{op_i^k\}_{i\in I,k\in K})$, and $C[p]$ is the program that results from replacing every dot $\bullet$ in C by p. It turns out that a semantics is compositional if and only if for all programs p_1, p_2 in $\mathscr{PL}$,

$$\mathscr{S}(p_1) = \mathscr{S}(p_2) \Leftrightarrow \text{for any context } C : \mathscr{S}(C[p_1]) = \mathscr{S}(C[p_2])$$

holds. Although this fact is folklore among the community of semantics we present a sketch of a proof. If the latter equivalence holds we define

$$\widehat{op}^k(S_1,\ldots,S_k) := \begin{cases} \mathscr{S}(op^k(p_1,\ldots,p_k)) & \text{for } p_i \in \mathscr{PL} \text{ with } \mathscr{S}(p_i) = S_i,\ 1\leqslant i\leqslant k, \\ \bot & \text{if } \nexists p_i : \mathscr{S}(p_i) = S_i \text{ for some } i,\ 1\leqslant i\leqslant k. \end{cases}$$

This definition is sound: if we have additional programs $p_1',\ldots,p_k'$ with $\mathscr{S}(p_i') = S_i$ for $1\leqslant i\leqslant k$, $\mathscr{S}(op^k(p_1,\ldots,p_k)) = \mathscr{S}(op^k(p_1',\ldots,p_k'))$ holds. This is easily seen as from $\mathscr{S}(p_i) = \mathscr{S}(p_i')$ one concludes

$$\mathscr{S}(op^k(p_1',\ldots,p_{i-1}', p_i, p_{i+1},\ldots,p_k)) = \mathscr{S}(op^k(p_1',\ldots,p_{i-1}', p_i', p_{i+1},\ldots,p_k)),$$

where $op^k(p_1',\ldots,p_{i-1}',\bullet,p_{i+1},\ldots,p_k)$ is a context for $1\leqslant i\leqslant k$. Chaining together these equations for $1\leqslant i\leqslant k$ yields $\mathscr{S}(op^k(p_1,\ldots,p_k)) = \mathscr{S}(op^k(p_1',\ldots,p_k'))$.

Only if: Suppose now that $\mathscr{S}$ is compositional. If $\mathscr{S}(C[p_1]) = \mathscr{S}(C[p_2])$ holds for all C we can choose $C = \bullet$, so $\mathscr{S}(p_1) = \mathscr{S}(p_2)$ holds. In case $\mathscr{S}(p_1) = \mathscr{S}(p_2)$ holds, we easily see that $\mathscr{S}(C[p_1]) = \mathscr{S}(C[p_2])$ is true for $C = \bullet$ and $C = p \in \mathscr{PL}$ as $\mathscr{S}(\bullet[p_1]) = \mathscr{S}(p_1) = \mathscr{S}(p_2) = \mathscr{S}(\bullet[p_2])$ and $\mathscr{S}(p[p_1]) = \mathscr{S}(p) = \mathscr{S}(p[p_2])$. We have to show that the equation also holds for a context $C = op^k(C_1,\ldots,C_k)$ where $C_1,\ldots,C_k$ are again contexts. By induction hypothesis we assume $\mathscr{S}(C_i[p_1]) = \mathscr{S}(C_i[p_2])$ for $1\leqslant i\leqslant k$. Then, by compositionality,

$$\begin{aligned} \mathscr{S}(C[p_1]) &= \mathscr{S}(op^k(C_1[p_1],\ldots,C_k[p_1])) = \widehat{op}^k(\mathscr{S}(C_1[p_1]),\ldots,\mathscr{S}(C_k[p_1])) \\ &= \widehat{op}^k(\mathscr{S}(C_1[p_2]),\ldots,\mathscr{S}(C_k[p_2])) = \mathscr{S}(op^k(C_1[p_2],\ldots,C_k[p_2])) \\ &= \mathscr{S}(C[p_2]). \end{aligned}$$

An alternative way of describing compositionality is to say that the semantic equivalence $\approx (p_1 \approx p_2 :\Leftrightarrow \mathscr{S}(p_1) = \mathscr{S}(p_2))$ is a congruence (with respect to the algebraic operations of $\mathscr{PL}$). To design a compositional semantics one can generally follow a continuation approach by setting $\mathscr{S} : \mathscr{PL} \to \mathscr{M}$ with $\mathscr{M} := \mathscr{C}on \to \mathscr{B}$, where $\mathscr{C}on$ is the class of contexts over the algebra of $\mathscr{PL}$ and $\mathscr{B}$ is a class of intended behaviours. Thus, $\mathscr{S}(p)$ is a mapping that describes the behaviour of p in any context: $\mathscr{S}(p) : \mathscr{C}on \to \mathscr{B}$, $\mathscr{S}(p)(C) := B(C[p])$. It is easily seen that any semantics defined

this way is compositional. We shall call a context U *universal* (for $\mathscr{C}on$ and B) if the following equation holds for all $p_1, p_2 \in \mathscr{PL}$: $B(U[p_1]) = B(U[p_2]) \Leftrightarrow \forall C \in \mathscr{C}on$: $B(C[p_1]) = B(C[p_2])$. If such a universal context U exists, $\mathscr{S}'(p) := B(U[p])$ obviously defines a semantics $\mathscr{S}' : \mathscr{PL} \to \mathscr{B}$ with the same semantic congruence $\simeq_{\mathscr{S}'}$ on $\mathscr{PL}$ as $\simeq_{\mathscr{S}}$. On the other hand, let B be any behaviour and C_0 any context such that $B(C_0[p_1]) = B(C_0[p_2])$ implies $B(p_1) = B(p_2)$ for all $p_1, p_2 \in \mathscr{PL}$. If $\mathscr{S}(p) := B(C_0[p])$ turns out to be compositional one can conclude that C_0 must be universal for B. A proof is completely trivial.

The story may be slightly more complicated in practice as a concrete programming language is not described by a single algebra but by a system of algebras, one for terms, another one for expressions, declarations, e.g., $\mathscr{PL}$ may also denote some dynamical systems, not necessary only programming languages. However, this does not change the connections described above between algebras, contexts, compositionality, and congruences.

In the case that $\mathscr{PL}$ involves some kind of parallelism, one distinguishes between *interleaving* and *true-concurrency* and between *linear time* and *branching time* semantics. These concepts are properties of $\mathscr{M}$. They describe whether the parallelism of $\mathscr{PL}$ can be seen directly in $\mathscr{M}$ (true-concurrency semantics) and whether the branching points for a non-deterministic behaviour are visible (branching time semantics). Thus, sets of strings (i.e., languages) are the classical example for a linear time interleaving semantics, while for a branching time interleaving semantics trees are normally used. To express true-concurrency partially ordered structures such as sets of pomsets in the linear time case and event-structures in the branching time case may be used. In connection with a semantics $\mathscr{S} : \mathscr{PL} \to \mathscr{M}$ Petri nets play a hybrid role. They are found both as $\mathscr{M}$ and as $\mathscr{PL}$. Thus, one talks about the "Petri net semantics" of $\mathscr{PL}$ if any $p \in \mathscr{PL}$ is interpreted by some Petri net. This follows the idea that the behaviour of a Petri net is sufficiently well understood. Thus, it suffices to interpret an object $p \in \mathscr{PL}$ by a Petri net to understand the true-concurrency behaviour of p. On the other hand, a Petri net may be considered as a directed bipartite graph which possesses a behaviour that is described by its token-game. In the first point of view, Petri nets are sufficiently well understood to play the role of $\mathscr{M}$ themselves. In the second case, Petri nets play the role of $\mathscr{PL}$ and their behaviour has to be defined in some appropriate mathematical structure $\mathscr{M}$. Both points of view have their own rights, depending on the level of abstraction one wishes to use. In this paper we shall adopt the second point of view, i.e., we investigate semantics $\mathscr{S} : \mathscr{PN} \to \mathscr{M}$ for the class $\mathscr{PN}$ of Petri nets. However, we immediately run into a serious difficulty as $\mathscr{PN}$ is not endowed with any canonical algebraic structure. Thus, the concept of compositionality does not apply and some people hesitate to talk about semantics in the absence of compositionality. Some classical concepts as the Petri net language $L(N)$ (see, e.g. [10–12, 18]) or the semi-language $S(N)$ of semi-words (i.e. only partially ordered words, see, e.g., [9, 14, 21]) of a Petri net N, however, play the role of such semantics without any compositionality. Also, there is no commonly accepted concept of a Petri net context. Therefore, we introduce such a concept of a context for Petri

nets and enrich the class of Petri nets with an algebraic structure. This will be such a simple generalization of Petri nets that its relevance will be more or less self-evident. Any Petri net is easily formed in this calculus from very trivial constants by applying very simple operations. Therefore, it is rather a surprise that several compositional semantics are easily definable to handle languages, step-languages, and pomset-languages for Petri nets.

1.2. Outline of a calculus for Petri nets and Petri net contexts

We investigate a very elementary Petri net algebra that follows canonically from the way one draws a graph of a Petri net. There will be nullary operations (i.e., constants) for single places and transitions, one binary operation $\|$ for the non-synchronized concurrent parallel product, and several unary operations for drawing arcs from places to transitions and from transitions to places and for putting tokens on places. However, if we define such a general algebra we encounter a serious problem. We can now define for any Petri net N and any single place p or transition t of N some context C such that $C[N]$ may inspect p and t itself. It is rather difficult for us to imagine an interesting behaviour such that the semantic congruence is not trivial in this case: If $\mathcal{S}(C[N_1]) = \mathcal{S}(C[N_2])$ has to hold for any context C (thus, also for contexts that inspect all places and transitions of N_1 and N_2), either $\mathcal{S}$ does not care about the places and transitions (and must thus identify too many intuitively different nets) or cares about the mere syntactic occurrence of places and transitions of N_1 and N_2, and so distinguishes too many syntactically different nets with, nevertheless, an intuitively equivalent behaviour. The solution is rather obvious. Some parts of a Petri net (places and transitions) have to be declared as private, others as public. A context must not add new arcs to private elements (or put tokens onto private places) but may communicate with the public elements. Thus, contexts must not inspect the private sections of a Petri net. We present an example of this idea in Fig. 1. The Petri nets N_1 and N_2 both behave like an asynchronous flip-flop-switch with two input gates (places 1 and 2) and three output gates (transitions a, b, and c). A stream of signals entering via input gate 1 passes a twice and then b once, and so on. A stream of signals entering via input gate 2 may pass to gate c but is blocked in the interval between a firing of b and a consecutive a. The places p_2 and p_4 and all transitions of N_1 and the places p_2 and p_6 and the transitions t_3, t_4, and t_5 of N_2 are public. All other parts belong to the inner structure that realizes the intended behaviour. We therefore will investigate Petri nets with *interfaces*. An interface is simply an ordered subset of places and transitions that are declared as public. Using this ordering we can talk about the ith interface place or ith interface transition. We use labelled Petri nets with a labelling function λ from the transitions to a set of actions or to the invisible action τ. Of course, public transitions must not be labelled by τ, as they are visible. However, we allow inner transitions (i.e., transitions that do not belong to the interface) to be labelled by either a visible or an invisible action, according to the standard of classical labelled Petri nets.

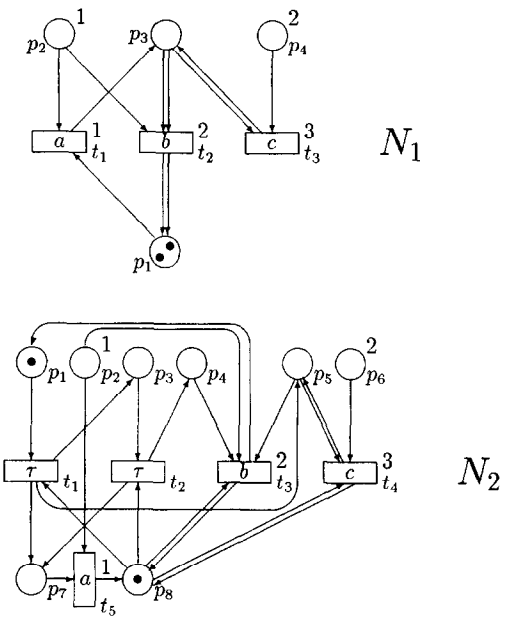

Fig. 1. Two Petri nets N_1, N_2 with the same interface (places 1, 2, transitions 1, 2, 3). Putting four tokens on interface place 1 and two tokens on interface place 2, e.g., leads to the same behaviour in both nets. In case of a language-behaviour, it leads to the language {accaba, aaccba, aabacc, acacba, acabac, aacbac}. No context that is connected to the interface places and transitions via additional arcs can distinguish N_1 and N_2.

For our algebra this implies that we deal with partial operations. Drawing an arc from place p_i to transition t_j is only allowed if p_i and t_j belong to the interface. The constants for a place or transition refer to public places and transitions. We therefore introduce additional hide-operations to simply remove objects from the interface and declare them as private. For a Petri net N, communication with a context C is achieved by adding some new arcs between the places and transitions of N and C. For a further discussion we refer to [16] where a restricted form of embedding a Petri net into a context is used in which only arcs from transitions to places may be added to connect a Petri net with its context.

Once we have endowed $\mathscr{PNI}$, the class of Petri nets with an interface, with an algebra we have a concept of contexts, $\mathscr{C}on$. The behaviours we are interested in are the language of N, the step-language of N, and the language of pomsets of N. We will be able to construct a (surprisingly trivial) universal context U for $\mathscr{PNI}$ for any of the three behaviours mentioned. In fact, one and the same U suffices for all three behaviours as our language and step-language semantics turn out to be (more or less) just special cases of the pomset semantics. We shall prove the universality of U indirectly by showing that $B(U[N])$ is compositional. Thus, there was only a small step left from the known non-compositional "semantics" of Petri nets to

algebraically smooth compositional semantics: define $\mathcal{S}(N)$ as $B(U[N])$ and not as $B(N)$.

The calculus that we present below is a generalization of the one used by Nielsen, Priese, Sassone [16] for (step-)languages. The main differences between [16] and this paper is that in [16] a Petri net could communicate with a context only by adding new arcs from transitions to places. In the new calculus we may add new arcs between places and transitions of the interface of a net and its context arbitrarily in any direction. A further, minor difference is that the Petri net calculus here can construct any Petri net from a finite number of atoms, two to be exact. We also investigate a compositional pomset semantics while [16] deals only with (step-)languages.

1.3. Connection to other calculi for Petri nets

Milner presents, in [15], an elementary calculus for Petri nets that allows for the introduction of single places and transitions and for their binding into larger nets via operations that declare the arc connections (using pre- and post-sets) and mark places, but no compositional semantics is given. This calculus is closely related to ours and we will see later how it can be expressed in terms of our calculus. In [3], Brown et al. introduce a language of nets with several operators chosen in such a way that a smooth compositional semantics can be given in terms of category theory with product, co-product, tensor product, etc. as semantic operators. While Milner's calculus describes the basic graphic structure of a Petri net, the calculus in [3] is on the other end of a line of abstractions: atoms may be arbitrarily complex Petri nets and the operators also reflect such abstract things as refinements. Using our figure of $\mathcal{S}: \mathcal{PL} \to \mathcal{M}$ for a semantics, Petri nets play the role of $\mathcal{PL}$ in both these calculi. The hybrid role of Petri nets as $\mathcal{PL}$ and $\mathcal{M}$ is clearly seen in [4], e.g., where the same Petri net algebra is used to specify both the structure of nets as well as their behaviours. However, Petri nets are mainly used as a semantics for other concurrent systems, see, e.g., [6–8, 13, 17, 22]. For this purpose it is not necessary to consider Petri nets as an algebra. Nevertheless, Petri nets as a semantic domain are also sometimes defined as a calculus. A well-known example is the Petri-Box calculus of Best. This calculus is used in [2] as a semantics for the concurrent language $B(PN)^2$ – however, in [13] also an operational semantics for the Petri-Box calculus is presented. Frequently, special cases of Petri nets are used as semantics for the behaviour of general Petri nets. Occurrence-nets and processes, [1], (they present mainly infinite unfoldings of Petri nets as a more dynamical model) may be seen in this way. As the set $Proc_N$ of all possible processes of a Petri net N is more or less equivalent to N (i.e., $Proc_{N_1} = Proc_{N_2}$ implies $N_1 = N_2$ up to minor syntactic differences in N_1 and N_2), processes are a semantics for Petri nets with a trivial semantic equivalence. However, one important use for them is that they can replace Petri nets which are used as a semantics for some other object. Thus, by processes one may change a Petri net semantics $\mathcal{S}: \mathcal{PL} \to \mathcal{PN}$, where $\mathcal{S}(p)$ is some Petri net, to a process semantics $\mathcal{S}': \mathcal{PL} \to Process$, where $\mathcal{S}(p)$ is replaced by $Proc_{\mathcal{S}(p)}$.

2. Basic Petri net notations

In this paper, Act denotes some fixed, infinite, countable set of *actions*. We frequently use a_i as names for actions. Let $M_\tau := M \cup \{\tau\}$, where τ is an *invisible* action not in Act, for any $M \subseteq Act$.

Definition 1 (*Labelled Petri nets with interface*). A finite labelled Petri net with an interface, *Petri net* for short, is a 7-tuple $N = (P_N, T_N, F_N, \lambda_N, s_N, \boldsymbol{p}, \boldsymbol{t})$, where

- P_N is a finite set of *places*, T_N is a finite set of *transitions*, and $P_N \cap T_N = \emptyset$,
- $F_N : (P_N \times T_N) \cup (T_N \times P_N) \to \mathbb{N}$ is a mapping defining directed (*multi*)*arcs* between transitions and places,
- $\lambda_N : T_N \to Act_\tau$ is a *labelling function*,
- $s_N : P_N \to \mathbb{N}$ is the *initial state* (or marking),
- $\boldsymbol{p} = (p_1, \ldots, p_m)$ is a vector of distinct places in P_N, the vector of ordered *interface places*,
- $\boldsymbol{t} = (t_1, \ldots, t_n)$ is a vector of distinct transitions in T_N, the vector of ordered *interface transitions*, such that $\lambda_N(t_i) \neq \tau$ for $1 \leqslant i \leqslant n$.

We denote the class of (finite, labelled) Petri nets with an interface by $\mathcal{PNI}$. A *classical Petri net* is a Petri net with an empty interface, $\boldsymbol{p} = ( ) = \boldsymbol{t}$. $(|\boldsymbol{p}|, |\boldsymbol{t}|)$ is called the *dimension* of N. Two Petri nets, $(P, T, F, \lambda, s, (p_1, \ldots, p_m), (t_1, \ldots, t_n))$, $(P', T', F', \lambda', s', (p'_1, \ldots, p'_m), (t'_1, \ldots, t'_n))$, of the same dimension are called equivalent if there is an isomorphism between (P, T, F, λ, s) and $(P', T', F', \lambda', s')$ which maps p_i to p'_i and t_j to t'_j, for $1 \leqslant i \leqslant m$ and $1 \leqslant j \leqslant n$.

Interface transitions must not be labelled by τ, but inner transitions (not belonging to the interface) may be labelled by some element of Act or by τ. Labels from Act_τ are standard in classical Petri net theory. We shall not distinguish equivalent Petri nets as they differ only by a renaming of the sets P and T. However, changing the order of the interface transitions or the interface places will result in a different Petri net. We shall assume that two Petri nets are always disjoint, which can be achieved by an appropriate renaming. We use a standard representation of a Petri net as a graph. Places are drawn as circles, transitions as bars, where the label in Act_τ is frequently written into the bar. The ith interface transition (place) is indicated by the number i attached to its bar (or circle, respectively).

Definition 2 (*Multisets, Steps, Step Languages*). A *multiset* μ on a set M is a function $\mu : M \to \mathbb{N}$; the *union* $\mu_1 \cup \mu_2$ of multisets μ_1, μ_2 is the multiset $\mu_1 + \mu_2$ such that $(\mu_1 + \mu_2)(e) = \mu_1(e) + \mu_2(e)$ for all $e \in M$. We use μ^M to denote the set of multisets on M. We also consider words of multisets where a multiset plays the role of a letter. We shall identify the empty word ε with the empty multiset, i.e., the function yielding 0 on all $e \in M$. A *step* is a multiset μ with finite support, i.e., $\{e \in M \mid \mu(e) > 0\}$ is finite. $STEP(M)$ denotes the set of all steps over M. $STEP^*(M)$ denotes the set of all (finite) words of steps over M. We embed M^* into $STEP^*(M)$ by identifying a single letter,

α, with the step $\{\alpha\}$ and a word $w=\alpha_1\ldots\alpha_k\in M^*$ with $\{\alpha_1\}\ldots\{\alpha_k\}\in STEP^*(M)$. A *step-language* (over M) is a subset L of $STEP^*(M)$. $STEP^*$ is an abbreviation of $STEP^*(Act)$.

Definition 3 (*Firing sequences, Petri net languages*). A *step* $X\in STEP(T_N)$ is *enabled* at a state $s\in\mu^{P_N}$ if $\sum_{t\in T_N}X(t)\cdot F_N(p,t)\leqslant s(p)$ for all $p\in P_N$; the firing of X at s leads to the state s', $s[X>s'$, where

$$s'(p)=s(p)+\sum_{t\in T_N}X(t)\cdot(F_N(t,p)-F_N(p,t)) \quad \text{for all } p\in P_N.$$

A *step-sequence* $X_1\ldots X_n$ *of* N is a word of steps such that $s_N[X_1>s_1$ and $s_i[X_{i+1}>s_{i+1}$ for $1\leqslant i<n$ for some states s_i. We also write $s_N[X_1\ldots X_n>s_n$. $F^{step}(N)$ denotes the set of all step-sequences of N. $X_1\ldots X_n$ is called *maximal* if $s_N[X_1\ldots X_n>s_n[X_{n+1}>s_{n+1}$ implies $X_{n+1}=\varepsilon$ and $s_n=s_{n+1}$. For a step X of N, let $\lambda_N(X)$ denote the multi-set of non-τ actions of X, i.e., for all $a\in Act$, $\lambda_N(X)(a)=\sum_{t\in\lambda_N^{-1}(a)}X(t)$. For a step-sequence $\chi=X_1\ldots X_n\in F^{step}(N)$ let $\lambda_N(\chi)$ be the word $\lambda_N(X_1)\ldots\lambda_N(X_n)$. The *step-language* $L^{step}(N)$ of N is the set $L^{step}(N):=\{\lambda_N(\chi)\mid\chi\in F^{step}(N)\}$. The step-sequences whose steps consist of at most one transition are called *firing-sequences*. $F(N)$ denotes the set of all firing-sequences of N. $L(N):=\lambda_N(F(N))$ is the standard *Petri net language* of N. $\mathscr{L}$ and $\mathscr{L}^{step}$ denote the class of all Petri net languages and step-languages, respectively.

Thus, Petri net languages belong to the interleaving semantic models as concurrency is neglected. Petri net step-languages are somewhere in the middle between interleaving and true-concurrency models: all actions of the same step are mutually concurrent – but not all concurrent actions have to be in one step. Thus, steps are non-deterministically chosen sets of mutually concurrent actions that are fired simultaneously. Of course, in (step-)languages the invisible actions τ are mapped to the empty word. Furthermore, step-languages of Petri nets are closed under prefixes and *unstepping*, where a prefix of a step-word is defined as for words and $\mathrm{Pref}\,L$ denotes the set of all prefixes of step-words in L. An unstepping splits a larger step into a series of smaller ones.

Definition 4 (*Unstepping and Linearization of Steps*). The operation unstep: $STEP^*\to 2^{STEP^*}$ of step-words is defined inductively for a step X and a step-word χ by

$$\mathrm{unstep}(X):=\{X_1X_2\ldots X_n\mid X_1+\cdots+X_n=X\},$$

$$\mathrm{unstep}(\chi X):=\mathrm{unstep}(\chi)\mathrm{unstep}(X).$$

The linearization lin of a step-word is its set of maximal unsteppings, i.e., $\mathrm{lin}: STEP^*\to 2^{Act^*}$ with $\mathrm{lin}(\chi):=\mathrm{unstep}(\chi)\cap Act^*$.

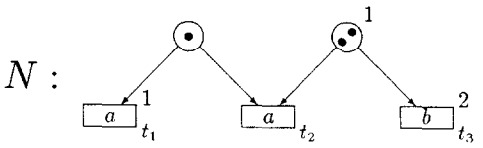

Fig. 2. An example of a Petri net.

Examples.

- $\text{unstep}(\{a,b\}\{a,c\}) = \{\{a,b\}\{a,c\},\{a\}\{b\}\{a,c\},\{b\}\{a\}\{a,c\},\{a,b\}\{a\}\{c\},$
$\{a,b\}\{c\}\{a\},\{a\}\{b\}\{a\}\{c\},\{b\}\{a\}\{a\}\{c\},$
$\{a\}\{b\}\{c\}\{a\},\{b\}\{a\}\{c\}\{a\}\}$
- We consider the Petri net N of Fig. 2. Obviously,

$$\begin{aligned} F^{step}(N) &= \{\varepsilon,\{t_1\},\{t_3\},\{t_1\}\{t_3\},\{t_3\}\{t_1\},\{t_1,t_3\},\{t_3\}\{t_3\},\{t_3,t_3\},\\ &\quad \{t_1\}\{t_3\}\{t_3\},\{t_3\}\{t_1\}\{t_3\},\{t_3\}\{t_3\}\{t_1\},\{t_1\}\{t_3,t_3\},\{t_3\}\{t_1,t_3\},\\ &\quad \{t_1,t_3\}\{t_3\},\{t_3,t_3\}\{t_1\},\{t_1,t_3,t_3\},\{t_2\},\{t_2\}\{t_3\},\{t_3\}\{t_2\},\{t_2,t_3\}\}\\ &= \text{Pref unstep}(\{t_1,t_3,t_3\},\{t_2,t_3\}),\\ L^{step}(N) &= \text{Pref unstep}(\{a,b,b\},\{a,b\}) = \text{Pref unstep}(\{a,b,b\}). \end{aligned}$$

We define causality structures and pomsets following [5].

Definition 5 (*Posets, causality structures, pomsets*). A *partially ordered set* (*poset*) is a pair $(X,<)$ of a set X and an irreflexive, transitive, anti-symmetric relation, a partial order $<$ on X. x is a *predecessor* of y if $x<y$ holds. x is a *direct predecessor* of y if $x<y$ holds and there is no z such that $x<z<y$ holds. In these cases, y is also called a (*direct*) *successor* of x. The *level* of x, $\text{lev}(x)$, is defined as $\sup\{n \mid \exists x_1,\ldots,x_n \in X\colon x_1<x_2<\cdots<x_n=x\}$, the *length* of $(X,<)$ by $\text{length}((X,<)) := \sup\{\text{lev}(x) \mid x\in X\}$.

A *causality structure* σ *over* Act is a triple $\sigma=(X,<,\ell)$ of a poset $(X,<)$ and a labelling function $\ell: X \to Act$, such that $\{x\in X \mid \text{lev}(x)\leqslant n\}$ is finite for all $n\in\mathbb{N}$ and $\text{lev}(x)<\omega$ for all $x\in X$.

Two causality structures σ, ρ are *isomorphic* if some bijection $\Psi: X_\sigma \to X_\rho$ exists such that $\Psi(x)<_\rho \Psi(y) \Leftrightarrow x<_\sigma y$ and $\ell_\rho \circ \Psi = \ell_\sigma$ holds for all $x,y\in X_\sigma$.

A *pomset* (*partially ordered multiset*) p is an isomorphism class of causality structures. $[\sigma]$ denotes the pomset with the causality structure σ as a representative. We call a pomset $[(X,<,\ell)]$ finite if X is finite. Let $POM^*(M)$ denote the set of all finite pomsets over some alphabet $M\subseteq Act$. Again, POM^* is an abbreviation of $POM^*(Act)$.

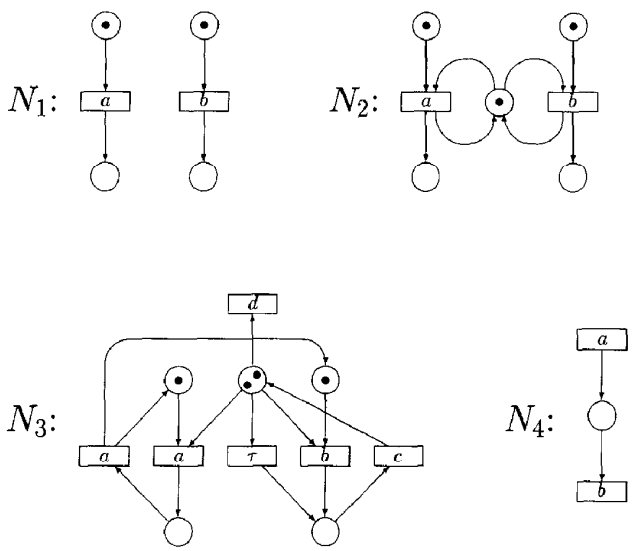

Fig. 3. Example nets for determining some pomset-behaviours.

We will only investigate finite pomsets in this paper. We graphically represent a pomset as a directed graph whose nodes are labelled by $\ell(X)$ and whose directed arcs represent the direct successor relation.

Example. The Petri nets N_i $(i=1,2,3)$ of Fig. 3 possess the following intuitive sets $B^{pom}(N_i)$ of pomset-behaviours, where $x \to y$ means that a transition labelled with x has to fire before y. $B^{pom}(N_1)=\{(\varepsilon),(a),(b),\binom{a}{b}\}$, $B^{pom}(N_2)=\{(\varepsilon),(a),(b),(a\to b),(b\to a)\}$. N_1 possesses a pomset $(a\ b)$ in its behaviour, N_2, however, possesses the two pomsets $(a\to b)$ and $(b\to a)$. N_3 has, amongst others, the following pomsets in its intuitive pomset-behaviour:

$$\binom{d}{d},\ \binom{a\to a}{d},\ \binom{b\to c\to c}{d},\ \binom{b\to c\to d}{c},\ \begin{pmatrix} & & c\to b\to c\to a\to a \\ a\to a & \nearrow & \end{pmatrix},$$

where for example $\binom{b\to c\to c}{d}$ can be prolonged. In $\binom{b\to c\to c}{d}$ the token created from the first c is used to fire τ and then c, which is why c depends on c. In $\binom{b\to c\to d}{c}$ a token independent from c is used to fire c (via τ).

We also call two nodes of a pomset graph *concurrent* if none is a predecessor of the other. For a formalization of the pomset-behaviour of a Petri net we follow the ideas of [1, 19] using an abstraction of processes.

Definition 6 (*Occurrence net, process*). An *occurrence net* is a triple $O=(B, E, F)$ with finite, disjoint sets of *conditions*, B, and *events*, E, and a relation $F\subseteq(B\times E)\cup$

$(E \times B)$ defining arcs between them. For any $b \in B$, $e_1, e_2 \in E$ we demand $(bFe_1 \wedge bFe_2 \Rightarrow e_1 = e_2)$ and $(e_1Fb \wedge e_2Fb \Rightarrow e_1 = e_2)$.

A *process of a Petri net*, $N = (P, T, F_N, \lambda_N, s_0, \boldsymbol{p}, \boldsymbol{t})$, is a triple (O, r, λ) of an occurrence net $O = (B, E, F)$, a mapping $r : B \cup E \to P \cup T$ with $r(B) \subseteq P$ and $r(E) \subseteq T$, canonically extended to multisets on $B \cup E$, and a labelling function $\lambda := \lambda_N \circ r$. r must preserve the structure of O, i.e., the conditions $r(F(\cdot, e)) = F_N(\cdot, r(e))$, $r(F(e, \cdot)) = F_N(r(e), \cdot)$ for $e \in E$, and $r(\{b \in B \mid F(\cdot, b) = \emptyset\}) = s_0$ must hold. By $Proc(N)$ we denote the set of all processes of a Petri net N. Additionally, a process (O, r, λ) is called *maximal* if no transition is enabled at its final state $s_f := r(\{b \in B \mid F(b, \cdot) = \emptyset\})$.

We consider a pomset of N as an abstraction $\Phi(\pi)$ of some process π of N where we drop all places and invisible actions.

Definition 7 (*Pomsets of a Petri net*). $\Phi : Proc(N) \to POM^*$ is defined as $\Phi((O, r, \lambda)) := [(X, <, \ell)]$ with $X := \{e \in E \mid \lambda(e) \neq \tau\}$, $< := F^+|_{X \times X}$, and $\ell := \lambda|_X$ for $O = (B, E, F)$. Let $L^{pom}(N) := \{\Phi(\pi) \mid \pi \text{ is a process of } N\}$ denote the pomset-language of N and $\mathscr{L}^{pom} := \{L^{pom}(N) \mid N \in \mathscr{PNI}\}$ the class of all Petri net pomset-languages.

We may sequentialize pomsets by introducing additional elements to their ordering relation.

Definition 8 (*Sequential ordering*). A pomset p is called *more sequential* than a pomset q if there are representations $p = [(X, <_p, \ell)]$ and $q = [(X, <_q, \ell)]$ with $<_q \subseteq <_p$. By $>_{seq}(q) := \{[(X, <, \ell)] \mid <_q \subseteq <\}$ we denote the set of all pomsets more sequential than q.

Obviously, we may consider words in Act^* as special pomsets (with a total, linear ordering). Analogously, a step-word $\chi = X_1 X_2 \ldots X_n$ is a special pomset where all elements in X_i are ordered before all elements in X_j, for $1 \leqslant i < j \leqslant n$, and all elements of the same step X_i are mutually concurrent, for $1 \leqslant i \leqslant n$. In this sense, we can write $Act^* \subseteq STEP^* \subseteq POM^*$.

Definition 9 (*Steps of a Pomset*). The set of steps, step(p), of a pomset p is defined by step: $POM^* \to 2^{STEP^*}$, with $\text{step}(p) := >_{seq}(p) \cap STEP^*$.

The following is obvious:

Lemma 1. *For any Petri net N:*
- $L^{step}(N) = \text{step}(L^{pom}(N))$,
- $L(N) = \text{lin}(L^{step}(N))$.

Definition 10 (*Algebraic operations on Pomsets*). Let $\Sigma, \Gamma \subseteq Act$. In language theory, a *homomorphism* $h : \Sigma \to \Gamma^*$ maps a letter into a word (with $h(w_1 w_2) = h(w_1)h(w_2)$ for all words $w_1, w_2 \in \Sigma^*$), a *fine homomorphism* $h : \Sigma \to \Gamma_\tau$ maps a letter into a letter or

the invisible action τ, and a *very fine homomorphism* (or *renaming*) $h: \Sigma \to \Gamma$ maps a letter into a letter. The application of a very fine homomorphism to a pomset is defined by $h(p) := [(X, <, h \circ \ell)]$ for $p = [(X, <, \ell)]$. For a fine homomorphism h, the newly invisible elements have to be deleted. Thus $h(p) := [(X', <', \ell')]$ with $X' := \{x \in X \mid h \circ \ell(x) \neq \tau\}$, $<' := <|_{X'}$, $\ell' := h \circ \ell|_{X'}$. The *inverse* of these mappings is defined as in set theory, i.e. $h^{-1}(K) := \{p \in POM^*(\Sigma) \mid h(p) \in K\}$ for $K \subseteq POM^*(\Gamma)$. The *shuffle* of two pomsets $p_i = [(X_i, <_i, \ell_i)]$, $i = 1, 2$, is defined by $p_1 \sqcup p_2 := [(X_1 \dot\cup X_2, <_1 \dot\cup <_2, \ell_1 \dot\cup \ell_2)]$. A *big shuffle*, $\bigsqcup$, is defined inductively on sets of pomsets by $\bigsqcup^0 L := \{[(\emptyset, \emptyset, \emptyset)]\}$, $\bigsqcup^{n+1} L := L \sqcup \bigsqcup^n L$, $\bigsqcup L := \bigcup_{n \geqslant 0} \bigsqcup^n L$. Further, $\bigsqcup p := \bigsqcup \{p\}$.

Note, as step-words $\chi \in STEP^*$ and words $w \in Act^*$ are special pomsets, $h(\chi)$ and $h(w)$ are already defined for fine and very fine homomorphisms. Of course, $h(w)$ coincides with the standard definition from language theory. However, for $\sqcup$ we need a new definition as for (step-)words the shuffle operator maps two words into a set of words. This set expresses all possible shufflings of the two words. Thus, $\chi_1 \sqcup \chi_2 := \text{step}(\hat\chi_1 \sqcup \hat\chi_2)$ and $w_1 \sqcup w_2 := \text{lin}(\text{step}(\hat w_1 \sqcup \hat w_2))$, where $\hat\chi_i$ and $\hat w_i$ denote χ_i and w_i considered as pomsets. Of course, this definition coincides with the standard shuffle-product for words. Furthermore, we can use the big shuffle, which has not been used in the literature before, also for languages of (step-)words. It turns out to be quite helpful. E.g., the pomset-language of the Petri net N_4 of Fig. 3 is easily expressed by $L^{pom}(N_4) = \bigsqcup\{(a), (a \to b)\}$. The language of N_4 is $L(N_4) = \bigsqcup\{\varepsilon, a, ab\} = \text{Pref} \bigsqcup ab$. Thus, $\bigsqcup ab$ is the Dyck-language over a and b, where a plays the role of "(" and b that of ")".

Definition 11 (*Algebraic operations on step-words*). Let χ, χ' denote step-words in $STEP^*$ and L a step-word language, i.e., $L \subseteq STEP^*$. We define the relation $\leqslant_\alpha$ for some letter $\alpha \in Act$ as the transitive closure of

$$\chi \leqslant^1_\alpha \chi' \quad :\Leftrightarrow \quad \exists \chi_1, \chi_2, \chi_3 \text{ step-words and } X_1, X_2 \text{ (possibly empty) steps s.t.}$$

$$\chi = \chi_1 (X_1 \cup \{\alpha\}) \chi_2 X_2 \chi_3 \quad \text{and} \quad \chi' = \chi_1 X_1 \chi_2 (X_2 \cup \{\alpha\}) \chi_3.$$

This relation may yield $\chi \leqslant^1_\alpha \chi'$ with $\text{length}(\chi) > \text{length}(\chi')$ in the case that X_1 is empty but X_2 is not. We define an alternative relation $\preccurlyeq^1_\alpha$ by

$$\chi \preccurlyeq^1_\alpha \chi' \quad :\Leftrightarrow \quad \chi \leqslant^1_\alpha \chi' \quad \text{and} \quad \text{length}(\chi) \leqslant \text{length}(\chi'),$$

and $\preccurlyeq_\alpha$ as its transitive closure. Let

$$\text{Min}_\alpha(L) := \{\chi \in L \mid \nexists \chi' \in L : \chi' \neq \chi \text{ and } \chi' \preccurlyeq_\alpha \chi\}.$$

In $\text{Min}_\alpha(L)$ all letters α are "transported as far to the left as possible in L" without introducing additional steps. In contrast to Min_α, in $\overleftarrow{\alpha}$ we shall shift letters α non-deterministically to the left, independent of whether the resulting words remain in L or not, or whether new steps are introduced:

$$\overleftarrow{\alpha}(L) := \{\chi \in STEP^* \mid \exists \chi' \in L : \chi \leqslant_\alpha \chi'\}.$$

Analogously, we can shift letters α to the right:

$$\overrightarrow{\alpha}(L) := \{\chi \in STEP^* \mid \exists \chi' \in L : \chi' \leqslant_\alpha \chi\}.$$

The L_1-*quotient* $L_1 \backslash L_2$ of L_2 is defined as

$$L_1 \backslash L_2 := \{u \in STEP^* \mid \exists w_1 \in L_1, w_2 \in L_2 : w_1 u = w_2\}.$$

The difference between $\leqslant_\alpha$ and $\preccurlyeq_\alpha$ is easily seen in an example. For three step-words consisting of actions a and y, $\{a\}\{a,y\}$, $\{a,y\}\{a\}$, and $\{y\}\{a\}\{a\}$, we can see that $\{y\}\{a\}\{a\} \leqslant_y^1 \{a,y\}\{a\} \leqslant_y^1 \{a\}\{a,y\}$ holds. The further the y moves to the left the "smaller" the word gets under $\leqslant_y^1$. Under $\preccurlyeq_y^1$ we have a different result. Still, $\{a,y\}\{a\} \preccurlyeq_y^1 \{a\}\{a,y\}$ holds, but $\{y\}\{a\}\{a\} \preccurlyeq_y^1 \{a,y\}\{a\}$ is false due to the higher number of steps in $\{y\}\{a\}\{a\}$.

As words in Act^* are special cases of step-words in $STEP^*$ we use the above definitions also for words. Obviously, those operations applied to words result in words or sets of words. E.g., for words u, v one gets $v \leqslant_\alpha u$ if there are words w_1, w_2, w_3 such that $v = w_1 \alpha w_2 w_3$ and $u = w_1 w_2 \alpha w_3$ hold. Note that $\leqslant_\alpha$ and $\preccurlyeq_\alpha$ coincide for words.

3. An elementary Petri net calculus

To define our algebra on Petri nets we shall use the following operators on $\mathscr{PNI}$, which are formally defined in Definition 12: There are three nullary operators, the constants 0, *place*, and *a-trans*, where 0 denotes the empty Petri net, *place* denotes the Petri net consisting of exactly one place (which is also the first and only interface place), and *a-trans* denotes a Petri net consisting of exactly one interface transition labelled with a. We use one binary operator, $\|$, the *parallel product* of two Petri nets without any arc-connection between them. Further, some unary operators are required, namely $arc_{i\to j}^{P\to T}$, which adds an arc from the ith interface place to the jth interface transition, $arc_{i\to j}^{T\to P}$, which adds an arc from the ith interface transition to the jth interface place, and $mark_s$, which adds tokens to some interface places. We choose $s \in \mu^{\mathbb{N}}$. $s(i)$ is the number of tokens to be added to the ith interface place. Of course, we identify s with the marking $\hat{s} \in \mu^{P_N}$ with $\hat{s}(p) = 0$ if $p \notin \boldsymbol{p}$ and $\hat{s}(p) = s(i)$ if p is the ith interface place. Furthermore, there is the unary operator $reorder_\pi$, which rearranges the order of the interface transitions and places as described by $\pi = (\pi_1, \pi_2) \in \gamma \times \gamma$; π is a pair of permutations, where $\gamma := \bigcup_{n \geqslant 0} \gamma_n$, and γ_n denotes the class of all permutations of n elements. $hide_i^P$ is another unary operator which hides the ith interface place from the interface – however, the place remains in the net, only the interface is changed. Besides there are $hide_i^T$, which hides the ith interface transition from the interface, and τ-$hide_i$, which works like $hide_i^T$ but additionally relabels the ith interface transition to τ. Finally, we use two operators $meld_{i,j}^P$ and $meld_{i,j;a}^T$ that merge two places or transitions into one. This new object takes position i in the interface. In

case of the $meld^T_{i,j;a}$-operator the newly created transition is labelled by the action symbol a.

These operations are not necessarily always defined. We consider, e.g., $hide^T_i(N)$ as undefined ($\perp$) if N does not possess an ith interface transition. The following definition explains formally the results of applying these operations to Petri nets.

Definition 12 (*Petri net operators*). The operators just mentioned are formally defined by the following operations (for some nets N, N_1, N_2).

$$0 := (\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,(\,),(\,)),$$

$$place := (\{p\},\emptyset,\emptyset,\emptyset,\emptyset,(p),(\,)),$$

$$a\text{-}trans := (\emptyset,\{t\},\emptyset,\lambda(t)=a,\emptyset,(\,),(t)) \quad \text{for } a \in Act,$$

$$N_1 \| N_2 := (P_1 \dot\cup P_2, T_1 \dot\cup T_2, F_1 \dot\cup F_2, \lambda_1 \dot\cup \lambda_2, s_1 + s_2, \boldsymbol{p}_1 \oplus \boldsymbol{p}_2, \boldsymbol{t}_1 \oplus \boldsymbol{t}_2),$$

$$\text{with } (u_1,\ldots,u_q) \oplus (v_1,\ldots,v_r) := (u_1,\ldots,u_q,v_1,\ldots,v_r),$$

$$arc^{P\to T}_{i\to j}(N) := (P_N, T_N, F', \lambda_N, s_N, \boldsymbol{p}_N, \boldsymbol{t}_N) \quad \text{for } 1 \leqslant i \leqslant |\boldsymbol{p}_N|,\ 1 \leqslant j \leqslant |\boldsymbol{t}_N|,$$

$$\text{with } F'(x,y) := \begin{cases} F_N(x,y) & \text{if } x \neq \boldsymbol{p}_N(i) \vee y \neq \boldsymbol{t}_N(j), \\ F_N(\boldsymbol{p}_N(i), \boldsymbol{t}_N(j)) + 1 & \text{otherwise,} \end{cases}$$

$$arc^{T\to P}_{i\to j}(N) := (P_N, T_N, F', \lambda_N, s_N, \boldsymbol{p}_N, \boldsymbol{t}_N) \quad \text{for } 1 \leqslant i \leqslant |\boldsymbol{t}_N|,\ 1 \leqslant j \leqslant |\boldsymbol{p}_N|,$$

$$\text{with } F'(x,y) := \begin{cases} F_N(x,y) & \text{if } x \neq \boldsymbol{t}_N(i) \vee y \neq \boldsymbol{p}_N(j), \\ F_N(\boldsymbol{t}_N(i), \boldsymbol{p}_N(j)) + 1 & \text{otherwise,} \end{cases}$$

$$mark_s(N) := (P_N, T_N, F_N, \lambda_N, s_N + s, \boldsymbol{p}_N, \boldsymbol{t}_N),$$

$$\text{for } s \in \mu^{\{p_1,\ldots,p_n\}} \text{ if } \boldsymbol{p}_N = (p_1,\ldots,p_n),$$

$$reorder_\pi(N) := (P_N, T_N, F_N, \lambda_N, s_N, \pi_1(\boldsymbol{p}_N), \pi_2(\boldsymbol{t}_N)),$$

$$\text{for } \pi = (\pi_1,\pi_2) \in \gamma_{|\boldsymbol{p}_N|} \times \gamma_{|\boldsymbol{t}_N|},$$

$$hide^P_i(N) := (P_N, T_N, F_N, \lambda_N, s_N, \Delta_i^{|\boldsymbol{p}_N|}(\boldsymbol{p}_N), \boldsymbol{t}_N) \quad \text{for } 1 \leqslant i \leqslant |\boldsymbol{p}_N|,$$

where $\Delta^n_i : M^n \to M^{n-1}$ for an arbitrary set M, is a projection that deletes the ith element of a vector of M^n: $\Delta^n_i((x_1,\ldots,x_n)) := (x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_n)$,

$$hide^T_i(N) := (P_N, T_N, F_N, \lambda_N, s_N, \boldsymbol{p}_N, \Delta_i^{|\boldsymbol{t}_N|}(\boldsymbol{t}_N)) \quad \text{for } 1 \leqslant i \leqslant |\boldsymbol{t}_N|,$$

$$\tau\text{-}hide_i(N) := (P_N, T_N, F_N, \lambda', s_N, \boldsymbol{p}_N, \Delta_i^{|\boldsymbol{t}_N|}(\boldsymbol{t}_N)) \quad \text{for } 1 \leqslant i \leqslant |\boldsymbol{t}_N|,$$

$$\text{with } \lambda'(t) = \begin{cases} \lambda_N(t) & \text{if } t \in T_N - \{\boldsymbol{t}_N(i)\}, \\ \tau & \text{if } t = \boldsymbol{t}_N(i). \end{cases}$$

$$meld^P_{i,j}(N) := (P_N - \{\boldsymbol{p}_N(j)\}, T_N, F', \lambda_N, s', \Delta_j^{|\boldsymbol{p}_N|}(\boldsymbol{p}_N), \boldsymbol{t}_N) \quad \text{for } 1 \leqslant i < j \leqslant |\boldsymbol{p}_N|,$$

$$\text{with } F'(x,y) := \begin{cases} F_N(\boldsymbol{p}_N(i), y) + F_N(\boldsymbol{p}_N(j), y) & \text{if } x = \boldsymbol{p}_N(i), \\ F_N(x, \boldsymbol{p}_N(i)) + F_N(x, \boldsymbol{p}_N(j)) & \text{if } y = \boldsymbol{p}_N(i), \\ F_N(x,y) & \text{otherwise,} \end{cases}$$

$$\text{and } s'(x) := \begin{cases} s_N(\boldsymbol{p}_N(i)) + s_N(\boldsymbol{p}_N(j)) & \text{if } x = \boldsymbol{p}_N(i), \\ s_N(x) & \text{otherwise,} \end{cases}$$

$$meld^T_{i,j;a}(N) := (P_N, T_N - \{\boldsymbol{t}_N(j)\}, F', \lambda', s_N, \boldsymbol{p}_N, \Delta_j^{|\boldsymbol{t}_N|}(\boldsymbol{t}_N))$$

$$\text{for } 1 \leqslant i < j \leqslant |\boldsymbol{t}_N| \wedge a \in Act,$$

$$\text{with } F'(x,y) := \begin{cases} F_N(\boldsymbol{t}_N(i), y) + F_N(\boldsymbol{t}_N(j), y) & \text{if } x = \boldsymbol{t}_N(i), \\ F_N(x, \boldsymbol{t}_N(i)) + F_N(x, \boldsymbol{t}_N(j)) & \text{if } y = \boldsymbol{t}_N(i), \\ F_N(x,y) & \text{otherwise,} \end{cases}$$

$$\text{and } \lambda'(x) := \begin{cases} a & \text{if } x = \boldsymbol{t}_N(i), \\ \lambda_N(x) & \text{otherwise.} \end{cases}$$

Definition 13 (*Derived operators*). We frequently use the following derived operators: $place^0 := 0$, $place^{k+1} := place \,\|\, place^k$, $\boldsymbol{a}_k\text{-}trans := a_1\text{-}trans \,\|\, \ldots \,\|\, a_k\text{-}trans$, where $\boldsymbol{a}_k$ denotes the vector $(a_1, \ldots, a_k)$. For a finite subset $I \subseteq \mathbb{N}$, $hide^P_I := hide^P_{I-max\,I} \circ hide^P_{max\,I}$. We define $hide^T_I$ and $\tau\text{-}hide_I$ analogously. For $\boldsymbol{v} = (i_1, \ldots, i_k)$ and $\boldsymbol{w} = (j_1, \ldots, j_k)$ let $arc^{P \to T}_{\boldsymbol{v} \to \boldsymbol{w}} := arc^{P \to T}_{i_1 \to j_1} \circ \cdots \circ arc^{P \to T}_{i_k \to j_k}$ and $arc^{T \to P}_{\boldsymbol{v} \to \boldsymbol{w}} := arc^{T \to P}_{i_1 \to j_1} \circ \cdots \circ arc^{T \to P}_{i_k \to j_k}$.

Example. The Petri nets of Fig. 4 are defined by the following operations. Here $\boldsymbol{n} := (1, \ldots, n)$ and $\pi := (id_n, \binom{1\ 2}{2\ 1})$ with $id_n = \binom{1\ 2\ \cdots\ n}{1\ 2\ \cdots\ n}$.

$$N_1 = hide^T_1 arc^{P \to T}_{1 \to 1}(place \,\|\, a\text{-}trans),$$

$$N_2 = mark_{\{2 \cdot \boldsymbol{p}(1)\}}(place^n),$$

$$N_3 = arc^{P \to T}_{\boldsymbol{n} \to \boldsymbol{n}}(place^n \,\|\, \boldsymbol{a}_n\text{-}trans)$$

$$= (arc^{P \to T}_{1 \to 1}(place \,\|\, a_1\text{-}trans)) \,\|\ldots\|\, (arc^{P \to T}_{1 \to 1}(place \,\|\, a_n\text{-}trans)),$$

$$N_4 = reorder_\pi hide^T_{\{3,\ldots,n\}}(N_3)$$

$$= (arc^{P \to T}_{(1,2) \to (2,1)}(place^2 \,\|\, (a_2, a_1)\text{-}trans))$$

$$\|(hide^T_{\{1,\ldots,n-2\}} arc^{P \to T}_{\boldsymbol{n-2} \to \boldsymbol{n-2}}(place^{n-2} \,\|\, (a_3, \ldots, a_n)\text{-}trans)), \text{ and}$$

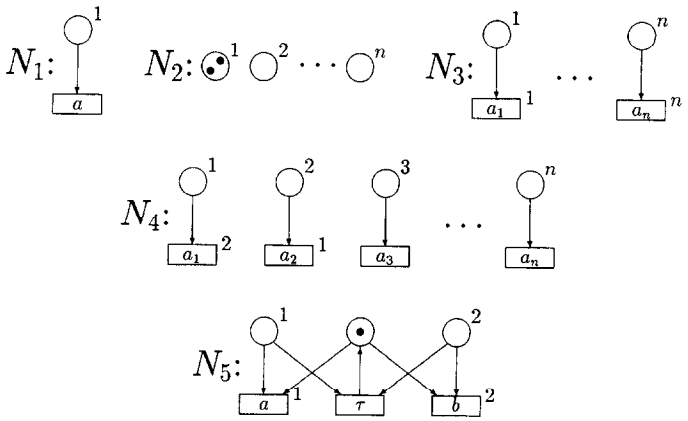

Fig. 4. Some examples of Petri nets with interface.

$N_5 = hide_2^P\, \tau\text{-}hide_2 mark_{\{1\cdot \mathbf{p}(2)\}} arc_{2\to 2}^{T\to P} arc_{(1,1,2,2,3,3)\to(1,2,1,3,2,3)}^{P\to T}(place^3 \,\|\, (a,c,b)\text{-}trans)$, where the second interface transition has to be labelled at the start (say by c) and is removed by $\tau\text{-}hide_2$ later.

It should be obvious that these operators suffice to define any Petri net in $\mathscr{PNI}$. In fact, they are redundant in order to allow smooth definitions. We shall use these operators to define a Petri net calculus $\mathscr{E}$ such that any net in $\mathscr{PNI}$ is definable in $\mathscr{E}$. In addition, we present several compositional semantics $\mathscr{S}$ for $\mathscr{E}$ that generalize some well-known semantics for classical Petri nets. In fact, we shall define several semantics $\mathscr{S}$ for $\mathscr{E}$ such that $\mathscr{S}(N)=L(N)$ or $\mathscr{S}(N)=L^{step}(N)$ or $\mathscr{S}(N)=L^{pom}(N)$ holds for classical Petri nets N without an interface.

It is quite a surprise that such interesting semantics are easily definable in a way that they become compositional for such a rich calculus as $\mathscr{E}$. If we draw a Petri net diagram for N using the above operators, we can define the compositional semantics $\mathscr{S}$ step by step, during the process of drawing the Petri net, in terms of the behaviours $\mathscr{S}(N')$ of already designed subparts N' of N. As an example, if we have already constructed N' and $S'=\mathscr{S}(N')$ and we design N as $arc_{i\to j}^{T\to P}(N')$ then $\mathscr{S}(N)=\mathscr{S}(arc_{i\to j}^{T\to P}(N'))$ is obtainable by an appropriate semantic operation solely on S', without any further knowledge of N' and N.

Definition 14 (*An elementary Petri net calculus* $\mathscr{E}$). $\mathscr{E}$ is a calculus over Act defined by

$$\forall a \in Act : \forall i,j \in \mathbb{N} : \forall s \in \mu^{\mathbb{N}} : \forall \pi \in \gamma \times \gamma :$$

$$PN :\equiv 0 \mid place \mid a\text{-}trans \mid PN \,\|\, PN \mid arc_{i\to j}^{P\to T}(PN) \mid arc_{i\to j}^{T\to P}(PN)$$
$$\mid mark_s(PN) \mid reorder_\pi(PN) \mid hide_i^P(PN) \mid hide_i^T(PN)$$
$$\mid \tau\text{-}hide_i(PN) \mid meld_{i,j}^P(PN) \mid meld_{i,j;a}^T(PN)$$

Definition 15 (*Petri net context*). A *Petri net context*, C, is an expression in $\mathcal{E}^\bullet$, where $\mathcal{E}^\bullet$ is the calculus $\mathcal{E}$ over Act enriched by a variable $\bullet$. $\mathcal{C} := \mathcal{E}^\bullet$ denotes the class of all contexts. Formally, $\mathcal{C}$ is defined by

$$\forall a \in Act : \forall i,j \in \mathbb{N} : \forall s \in \mu^{\mathbb{N}} : \forall \pi \in \gamma \times \gamma :$$
$$C :\equiv \bullet \mid 0 \mid place \mid a\text{-}trans \mid C \parallel C \mid arc_{i\to j}^{P\to T}(C) \mid arc_{i\to j}^{T\to P}(C)$$
$$\mid mark_s(C) \mid reorder_\pi(C) \mid hide_i^P(C) \mid hide_i^T(C)$$
$$\mid \tau\text{-}hide_i(C) \mid meld_{i,j}^P(C) \mid meld_{i,j;a}^T(C)$$

For $C \in \mathcal{C}$ and $N \in \mathcal{PNI}$ we denote by $C[N]$ the Petri net that results from replacing each $\bullet$ in C by N.

Definition 16 (*Universal context*). Let $\mathcal{X} := \{x_i^+, x_i^- \mid i \in \mathbb{N}\} \subseteq Act$ and $\mathcal{Y} := \{y_i \mid i \in \mathbb{N}\} \subset Act$ be sets of special action names solely used by the *universal context* $U_{m,n}$ (*of dimension* (m,n)), for $m,n \in \mathbb{N}$, where $U_{m,n}$ is defined as the $\mathcal{C}$-expression

$$U_{m,n} := arc_{m+n\to m+n}^{T\to P} hide_m^T arc_{m\to m}^{P\to T}(U_{m,n}^{\mathcal{X}} \parallel \bullet \parallel U_{m,n}^{\mathcal{Y}})$$

with

$$U_{m,n}^{\mathcal{X}} := x_m^+\text{-}trans \parallel x_m^-\text{-}trans,$$

and

$$U_{m,n}^{\mathcal{Y}} := mark_{\{1,\ldots,n\}} hide_{y_n}^T arc_{n\to n}^{P\to T}(place^n \parallel y_n\text{-}trans).$$

For a Petri net N of dimension (m,n) we use $U[N]$ as an abbreviation for $U_{m,n}[N]$.

Fig. 5 presents an example of a Petri net N of dimension (m,n) embedded into $U_{m,n}$. The ith interface transition of N becomes the $(i+m)$th in $U[N]$, the jth interface place of U the $(j+m)$th in $U[N]$. In the next chapter we shall prove that U is indeed a universal context in the sense of Section 1.

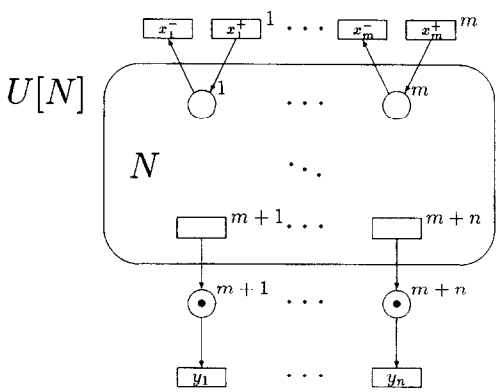

Fig. 5. Some net N with dimension (m,n) embedded into its universal context $U_{m,n}$.

4. Pomset semantics

Definition 17 (*Pomset semantics*). The *pomset semantics* $\mathcal{S}^{pom}: \mathcal{PNI} \to \mathcal{L}_0^{pom}$ is defined by

$$\mathcal{S}^{pom}(N) := L_0^{pom}(U[N]),$$

with

$$L_0^{pom}(U[N]) := \{ p \in L^{pom}(U[N]) \mid \exists O = (B,E,F), \text{ an occurrence net,}$$
$$\exists \pi = (O,r,\lambda) \in Proc(U[N]), \text{ a process of } U[N], \text{ such that}$$
$$p = \Phi(\pi) \text{ and } r(\{b \in B \mid F(b,\cdot) = \emptyset\}) \subseteq P_N \text{ holds}\},$$

$$\mathcal{L}_0^{pom} := \{L_0^{pom}(U[N]) \mid N \in \mathcal{PNI}\}.$$

As $r(\{b \in B \mid F(\cdot,b) = \emptyset\})$ denotes the initial marking of a process, $r(\{b \in B \mid F(b,\cdot) = \emptyset\})$ is its final marking. Thus, the requirement $r(\{b \in B \mid F(b,\cdot) = \emptyset\}) \subseteq P_N$ reads that finally no tokens are left in U. $L_0^{pom}(U[N])$ is thus a terminal Petri net pomset-language (see, e.g. [11] for terminal Petri net languages) where at the end all context-places are emptied. We will see the necessity of this terminal condition later. It is important to note that $\mathcal{S}^{pom}(N) = L^{pom}(N)$ for a classical Petri net without an interface. Thus, $\mathcal{S}^{pom}$ is a conservative generalization of the pomset-languages of classical Petri nets to Petri nets with an interface.

Theorem 1 (Compositionality of the pomset semantics). *$\mathcal{S}^{pom}$ is compositional for $\mathcal{E}$.*

Proof. We present for each k-ary operator op^k in $\mathcal{E}$ a k-ary operation $\widehat{op}^k : (\mathcal{L}_0^{pom})^k \to \mathcal{L}_0^{pom}$ such that for all $N_1,\ldots,N_k \in \mathcal{PNI}$ the following equation holds.

$$\mathcal{S}^{pom}(op^k(N_1,\ldots,N_k))) = \widehat{op}^k(\mathcal{S}^{pom}(N_1),\ldots,\mathcal{S}^{pom}(N_k)). \tag{1}$$

The following operations $\widehat{\ }$ obviously fulfill Eq. (1).

- $\widehat{0} := \{(\varepsilon)\} = \{[(\emptyset,\emptyset,\emptyset)]\}$.
- $\widehat{place} := L_0^{pom}(U[place]) = L^{pom}(\boxed{x_1^+} \overset{1}{\longrightarrow} \bigcirc \overset{1}{\longrightarrow} \boxed{x_1^-}) = \bigsqcup\{(x_1^+ \to x_1^-),(x_1^+)\}$. I.e., any pomset in $\widehat{place}$ is a graph of some unconnected pairs of nodes labelled by x_1^+ and x_1^- with an arc from x_1^+ to x_1^- in each pair plus some single nodes labelled by x_1^+. This also means that we allow for auto-concurrency in our model.
- $\widehat{a\text{-}trans} := L_0^{pom}(U[a\text{-}trans]) = L_0^{pom}(\boxed{a} \overset{1}{\longrightarrow} \odot \overset{1}{\longrightarrow} \boxed{y_1}) = (y_1) \sqcup \bigsqcup(a \to y_1)$.
- If N_1 and N_2 are Petri nets of dimension (m_i,n_i), $i = 1,2$, then $N_1 \| N_2$ is of dimension (m_1+m_2, n_1+n_2) and the ith interface transition of N_2 becomes the n_1+ith interface transition of $N_1 \| N_2$. Thus, we have to rename the labels $x_1^{\pm},\ldots,x_{m_2}^{\pm}$, $y_1,\ldots,y_{n_2}$ of $U_{m_2,n_2}[N_2]$ to $x_{i+m_1}^{\pm}$, y_{i+n_1} for $U_{m_1+m_2,n_1+n_2}[N_1 \| N_2]$, which can be done by a fine

homomorphism h_{L_1}. A compositional semantics has to operate on $\mathscr{L}_0^{pom}$, i.e. without knowing the Petri nets that gave rise to a particular language in $\mathscr{L}_0^{pom}$. But fortunately, the universal context allows us to detect the dimension (m_1, n_1) from the language L_1 by setting $m_1 := \max\{j \mid (x_j^+) \in L_1\}$, $n_1 := \max\{j \mid (y_j) \in L_1\}$. This is why we had to put the tokens on the places of U, see Fig. 5: Any y_j for $j \leqslant n_1$ is firable and thus appears as a pomset in L_1.

$$L_1 \,\widehat{\|}\, L_2 := L_1 \amalg h_{L_1}(L_2),$$

with

$$h_{L_1} : a \mapsto \begin{cases} x^+_{i+m_1} & \text{if } a = x_i^+ \text{ and } m_1 = \max\{j \mid (x_j^+) \in L_1\}, \\ x^-_{i+m_1} & \text{if } a = x_i^- \text{ and } m_1 = \max\{j \mid (x_j^+) \in L_1\}, \\ y_{i+n_1} & \text{if } a = y_i \text{ and } n_1 = \max\{j \mid (y_j) \in L_1\}, \\ a & \text{otherwise.} \end{cases}$$

As we can detect the dimension from the language we can always check if the parameters of a semantic operation are in range and just define the semantics to be $\perp$ if the evaluation of the Petri net expression would lead to an undefined net. For correct parameters, the remaining operations work as follows:

- $\widehat{reorder}_\pi(L) := \hat{\pi}(L)$, with

$$\hat{\pi} : a \mapsto \begin{cases} x^+_{\pi_1(i)} & \text{if } a = x_i^+ \in \mathscr{X}, \\ x^-_{\pi_1(i)} & \text{if } a = x_i^- \in \mathscr{X}, \\ y_{\pi_2(j)} & \text{if } a = y_j \in \mathscr{Y}, \\ a & \text{otherwise,} \end{cases}$$

  where $\pi = (\pi_1, \pi_2)$ is the reordering of the interface.
- For a $\widehat{mark}$ operation, we must be able to put a token onto an interface place or, equivalently, allow for an invisible firing of the corresponding x_i^+-transition. This can obviously be done by relabelling one such x_i^+ in every pomset to τ. Therefore, we define delete$_s$: $POM^* \to 2^{POM^*}$ for $s = (s(1), \ldots, s(n)) \in \mathbb{N}^n$ by $q \in \text{delete}_s(p) :\Leftrightarrow$ q is obtained from p by deleting $s(i)$ arbitrary nodes labelled with x_i^+, for $1 \leqslant i \leqslant n$. Thus, $\widehat{mark}_s(L) := \text{delete}_s(L)$.

For an intuitive meaning of the x_i^+- and x_i^--labels, notice that applying a $\widehat{mark}_{1 \cdot p(i)}$ operation to a pomset of the form $(x_i^+ \to x_i^-)$ (as produced by $\widehat{place}$) will yield a single x_i^--node. We therefore may interpret such an x_i^--node as the availability of a token on the ith interface place. We gain access to this token by removing the x_i^--node and thereby inhibiting this firing of the x_i^--transition. As the x_i^--transition consumes one token, we now have an additional token on the ith interface place. We might note here that an x_i^--node has no successors as it does not produce any tokens, but it may have predecessors. The predecessors of an x_i^--node are the labels of those transitions that were necessary for the production of the corresponding token. Dually, an x_i^+-node may have successors but no predecessors and represents the necessity of producing a

token. An x_i^+-node precedes all labels of transitions that depend on the production of this token.

- After having applied an $arc_{i\to j}^{T\to P}$, the ith interface transition produces one extra token on the jth interface place for each firing. We therefore replace a token produced by the x_j^+-transition with one produced by the ith interface transition. Any transition consuming this token is now causally dependent on the ith interface transition. Since such a transition was formerly dependent on some x_j^+ we just have to introduce new arcs from the ith interface transition to any successor of some x_j^+ and, afterwards, remove the x_j^+. We have to make sure here that the x_j^+ chosen is not a predecessor of the label of the ith interface transition we handle at that moment, otherwise our algorithm would introduce cycles. To detect this ith interface transition t_i in our semantic domain (we are not operating on Petri nets here) we exploit a property of L_0^{pom}: no token is allowed to stay within U. As any action-label of the ith interface transition t_i in L results from a firing of t_i the y_i-transition must fire after t_i to finally remove the corresponding token from U. As a consequence, the action-labels of t_i in L are precisely the direct predecessors of y_i-actions in the pomsets. Thus, we define $\widehat{arc}_{i\to j}^{T\to P}(L) := F_{j,i}(L)$, where $F_{j,i}$ operates on a pomset p as follows.

  For all nodes v in p with label y_i do
  for all direct predecessors v' of v in p do
  {there is at most one v' with $v' \to v$ in p as it requires one token to fire a y-transition of a universal context}
  choose one node v'' in p labelled by x_j^+ that is no predecessor of v' ($v'' \not< v'$)
  for all direct successors v''' of v'' {$v'' \to v'''$ holds in p} do
  add an arrow from v' to v''' in p od
  delete v''
  od
  od

  Fig. 6 shows an application of $F_{j,i}$.
- The $\widehat{arc}_{i\to j}^{P\to T}$-operator works analogously to the $\widehat{arc}_{i\to j}^{T\to P}$-operator. After having applied an $arc_{i\to j}^{P\to T}$, the jth interface transition needs one extra token on the ith interface place for each firing. Such an additional token may be available in form of an x_i^--node. By removing such a possible x_i^--node we inhibit the corresponding firing of the x_i^--transition giving us one extra token on the ith interface place. When we use this token for some firing of the jth interface transition that firing must be made causally dependent on the predecessors of the former x_i^--node. Our algorithm works like this: We introduce new arrows from any direct predecessor of the x_i^--node to the node of the jth interface transition and remove the x_i^--node. Again, we have to guarantee that no cycles are introduced. Therefore, we must not use an x_i^--node that is a successor of the label of the jth interface transition handled at that moment. Of course, labels of the jth interface transition in a pomset can again be identified as direct predecessors of y_j-labels. Thus, we define $\widehat{arc}_{i\to j}^{P\to T}(L) := G_{i,j}(L)$, where $G_{i,j}$ operates on a pomset p as follows.

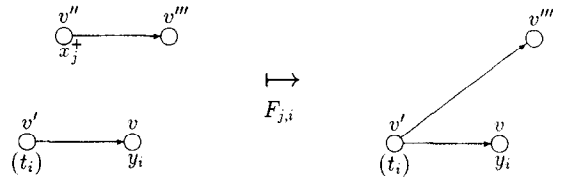

Fig. 6. $F_{j,i}$ introduces from any direct predecessor of any y_i-node a new arc to the direct successor of one (arbitrarily chosen) x_j^+-node such that no loops are introduced and eliminates the x_j^+-node.

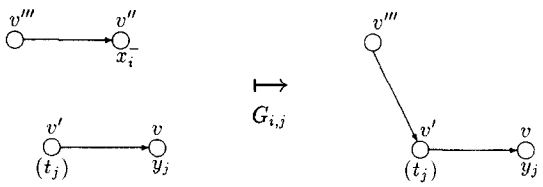

Fig. 7. $G_{i,j}$ introduces a new arc from each direct predecessor of one (arbitrarily chosen) x_i^--node to the direct predecessor of an y_j-node such that no loops are introduced and then eliminates the x_i^--node.

For all nodes v in p with label y_i do
for all direct predecessors v' of v in p do
choose one node v'' in p labelled by x_j^- that is no successor of v' $(v' \not< v'')$
for all direct predecessors v''' of v'' $\{v''' \to v''$ holds in $p\}$ do
add an arrow from v''' to v' in p od
delete v''
od
od

Fig. 7 shows an application of $G_{i,j}$.

- $hide_i^P$ removes the ith interface place from the interface. Thus, $\mathcal{S}^{pom}(hide_i^P(N)) = L_0^{pom}(U_{m-1,n}[hide_i^P(N)]) = L_0^{pom}(\text{“}hide_i^P\text{”}U_{m,n}[N])$ holds informally for $\dim N=(m,n)$, where “$hide_i^P$” has to change $U_{m,n}$ into $U_{m-1,n}$. As a consequence, we have to drop all pomsets in $\widehat{hide}_i^P(L_0^{pom}(U_{m,n}[N]))$ that use the $x_i^{\pm}$-transitions of $U_{m,n}$ and apply a renaming afterwards.

$$\widehat{hide}_i^P(L) := h_{x_i}(L \cap POM^*(Act - \{x_i^+, x_i^-\})),$$

where

$$h_{x_i} : a \mapsto \begin{cases} x_{j-1}^+ & \text{if } a = x_j^+ \text{ and } j > i, \\ x_{j-1}^- & \text{if } a = x_j^- \text{ and } j > i, \\ a & \text{otherwise.} \end{cases}$$

- If we want to hide the ith interface transition of N we have to remove the y_i-transition and the corresponding place from U. As this y_i-transition has no arc into

any further place of $U[N]$ we simply rename it to τ.

$$\widehat{hide}_i^T(L) := \bar{h}_{y_i}(L),$$

where

$$\bar{h}_{y_i} : a \mapsto \begin{cases} y_{j-1} & \text{if } a = y_j \text{ and } j > i, \\ \varepsilon & \text{if } a = y_i, \\ a & \text{otherwise.} \end{cases}$$

- For τ-$hide_i$ it is not sufficient to rename y_i to τ but we also have to relabel the ith interface transition t_i by τ. The labels of the ith interface transition in a pomset can again be identified as direct predecessors of y_i-labels. Thus, one gets $\tau\text{-}\widehat{hide}_i(L) := \bar{h}_{y_i} \circ G_{y_i}(L)$, where $G_{y_i}(p)$ deletes, in a pomset p, the direct predecessor node of any node labelled with y_i.
- The effect of a $meld_{i,j}^P$ operation in the semantics is that tokens are allowed to flow from the ith to the jth interface place and vice versa in an unobservable manner. Clearly, this can be realized by introducing two transitions, one consuming a token from the ith interface place and producing one on the jth interface place, the other one with the reverse effect. For an unobservable flow of tokens these two transitions have to be relabelled to τ. Finally, we have to remove the jth interface place from the interface by a $hide^P$ operation. As we can express the effect of $meld^P$ by operations already proven compositional, $meld^P$ is compositional itself.
- The $meld_{i,j;a}^T$-operator forces the ith and jth interface transition to fire simultaneously. This can easily be expressed in the semantics. For each node of the ith interface transition in any pomset p there must be exactly one node of the jth interface transition in p. The two nodes must not be causally ordered as they are to occur simultaneously. We replace both nodes by a new node labelled with a. This node a will have the predecessors and successors of the two original nodes as its predecessors and successors, respectively. We notice that it has direct successors y_i and y_j now. The node a is the label of the new ith interface transition, so we remove the y_j. We define $\widehat{meld}_{i,j;a}^T(L) := \bar{h}_{y_j} M_{i,j;a}(L)$, where $M_{i,j;a}$ generates a new pomset from a given pomset p as follows.

```
For each node v with a direct successor v' labelled by y_i in p do
  select a node v'' with a direct successor v''' labelled by y_j in p
  such that v and v'' are concurrent
    reject p if there is no such node v''
    for each direct predecessor u of v'' do
      add an arrow from u to v
    od
    for each direct successor u' of v'' do
      add an arrow from v to u'
    od
```

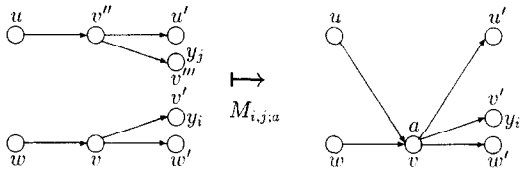

Fig. 8. $M_{i,j;a}$ repeatedly selects both a node v belonging to the ith interface transition and a node v'' of the jth interface transition. The node v'' and its y_j-successor are removed. The node v inherits all arrows of v''. $M_{i,j;a}$ only works if each chosen pair (v, v'') is concurrent and the number of y_i's and y_j's in the pomset is balanced.

delete v'' and v'''
relabel v by a
od
reject p if there remains a node v'' with a direct successor labelled by y_j in p

Fig. 8 shows the effect of $M_{i,j;a}$ on a pomset. □

This proves $\mathcal{S}^{pom}(N) := L_0^{pom}(U[N])$ to be compositional (with respect to $\mathcal{E}$). As a consequence, U is also universal for the L_0^{pom}-behaviour (see Section 1). From this proof it should be transparent why we use tokens on the interface places of U and why the fairness-condition for L_0^{pom} (that all enabled y_j-transitions must fire) is required. We need the tokens to detect the dimension of a Petri net N with $L = L_0^{pom}(U[N])$ from L. The dimension is important for a correct "game" with names in $\mathcal{X}$ and $\mathcal{Y}$. The reason we defined $\mathcal{S}^{pom}(N)$ by $L_0^{pom}(U[N])$ instead of $L^{pom}(U[N])$ is to ensure that any y_j follows immediately after the label of the jth interface transition of N in any pomset of N. In a prefix-closed language (as $L(U[N])$, in contrast to $L_0(U[N])$) this fails for prefixes where y_j's are cut off.

5. Step and word semantics

In the previous chapter we have interpreted the meaning of a Petri net by a set (a language) of pomsets. With the help of a universal context we could define a compositional pomset semantics with a very simple proof for compositionality. The reason for this is that pomsets allow us to describe the behaviour of a Petri net so "truly" that our syntactic operations on Petri nets turn easily into algebraic operators for the semantics. E.g., hiding the ith interface transition and relabelling it to τ is trivially reflected by deleting all y_i labels and their direct predecessors. We now transform this technique to step-languages and classical interleaving languages. I.e., the behaviour of a Petri net will now be interpreted by a language of step-words and words. The idea is that a single pomset p may be sufficiently well described by its set of all possible step-words, step(p), or by its set of all possible linearizations, lin(step(p)). Thus, if for all semantic operators $\widehat{op}^k : (\mathcal{L}_0^{pom})^k \to \mathcal{L}_0^{pom}$ of the proof of

Theorem 1 the equations $\mathrm{step}(\widehat{op}^k(L_1,\ldots,L_k)) = \mathrm{step}(\widehat{op}^k(\mathrm{step}(L_1),\ldots,\mathrm{step}(L_k)))$ and $\mathrm{lin}\circ\mathrm{step}(\widehat{op}^k(L_1,\ldots,L_k)) = \mathrm{lin}\circ\mathrm{step}(\widehat{op}^k(\mathrm{lin}\circ\mathrm{step}(L_1),\ldots,\mathrm{lin}\circ\mathrm{step}(L_k)))$ should hold we would get step- and language semantics as trivial special cases of our pomset semantics: Set $\mathcal{S}^{step}(N) := \mathrm{step}\,\mathcal{S}^{pom}(N)$, $\mathcal{S}(N) := \mathrm{lin}\circ\mathrm{step}\,\mathcal{S}^{pom}(N)$ and get compositionality for free from both equations. E.g., we might conclude:

$$\begin{aligned}\mathcal{S}^{step}(op^k(N_1,\ldots,N_k)) &= \mathrm{step}(\mathcal{S}^{pom}(op^k(N_1,\ldots,N_k)))\\ &= \mathrm{step}(\widehat{op}^k(\mathcal{S}^{pom}(N_1),\ldots,\mathcal{S}^{pom}(N_k)))\\ &= \mathrm{step}\circ\widehat{op}^k(\mathrm{step}\circ\mathcal{S}^{pom}(N_1),\ldots,\mathrm{step}\circ\mathcal{S}^{pom}(N_k))\\ &= \mathrm{step}\circ\widehat{op}^k(\mathcal{S}^{step}(N_1),\ldots,\mathcal{S}^{step}(N_k)).\end{aligned}$$

However, both equations do not hold, but the situation is only slightly more complicated. Suppose again that we want to hide the ith interface transition and relabel it to τ. Thus, if we find $\binom{a\to y_j}{b}$ in some pomset p, we simply replace this part of p by (b). However, $\mathrm{lin}\circ\mathrm{step}(p)$ $\binom{a\to y_j}{b}$ turns into the set $ay_j \sqcup\!\sqcup\, b = \{ay_jb, aby_j, bay_j\}$. Thus, the direct predecessor a of y_j in p usually cannot be detected in a single word of $\mathrm{lin}\circ\mathrm{step}(p)$ – but is detectable from the whole set $\mathrm{lin}\circ\mathrm{step}(p)$. This may easily be done with the help of a Min operation (compare Definition 11). In the above example we first have to apply Min_{y_j} to $\mathrm{lin}\circ\mathrm{step}(p)$, resulting in $\mathrm{Min}_{y_j}(ay_j \sqcup\!\sqcup\, b) = \{ay_jb, bay_j\}$, which yields the correct direct predecessor of y_j if p. In step-languages, the Min_{y_j} operation allows for the detection of steps in which the jth interface transition is contained. A further problem arises when we need to identify the label of a transition and not only its position. In a step-word $\{a,b\}\{y_j\}$, as it may be produced by a Min_{y_j} operation, we can see in which step the label of the jth interface transition appears, but we cannot identify the label itself. (It may be the a or the b here.) Luckily, step-languages of Petri nets are closed under unstepping. We can always find a step-word in which the label of the jth interface transition is the only label in its step. Then, of course, we know the label and can also identify it in step-words as $\{a,b\}\{y_j\}$. Playing with this and similar tricks on the whole set of steps or linearizations of a single pomset, most of the semantic operators on pomsets are easily translated to new operators on languages of steps or words.

Definition 18 (*Step semantics* $\mathcal{S}^{step}$). $\mathcal{S}^{step} : \mathcal{PNI} \to \mathcal{L}_0^{step}$ is defined by

$$\mathcal{S}^{step}(N) := L_0^{step}(U[N]),$$

with

$$\begin{aligned}L_0^{step}(U[N]) := \{\lambda_{U[N]}(\chi) \mid \chi \in F^{step}(U[N]) \text{ and } \exists s(s_{U[N]}[\chi > s \text{ and no } y_j,\\ 1 \leqslant j \leqslant |t_N|, \text{ is enabled in } s)\}\end{aligned}$$

$$\mathcal{L}_0^{step} := \{L_0^{step}(U[N]) \mid N \in \mathcal{PNI}\}.$$

Theorem 2 (Compositionality of $\mathcal{S}^{step}$). *$\mathcal{S}^{step}$ is compositional for $\mathcal{E}$.*

Proof. Obviously, $\mathcal{L}_0^{step} = \text{step}(\mathcal{L}_0^{pom})$ and $\mathcal{S}^{step} = \text{step} \circ \mathcal{S}^{pom}$ hold. We exploit the compositionality of $\mathcal{S}^{pom}$ to prove the compositionality of $\mathcal{S}^{step}$. There are some (k-ary) operators Op^k used in the proof of Theorem 1 with the property

$$\text{step}(Op^k(L_1,\ldots,L_k)) = \text{step}(Op^k(\text{step}(L_1),\ldots,\text{step}(L_k))) \tag{2}$$

for all $L_1,\ldots,L_k \in \mathcal{L}_0^{pom}$. For these operators we simply use $\text{step} \circ Op^k$ in $\mathcal{S}^{step}$. Such operators are $\coprod$, $\sqcup\!\!\sqcup$, h_L, $\hat{\pi}$, h_x, $\cap$, $\bar{h}_y$, $h_{i,j}$, and delete_s. However, Eq. (2) fails for G_y, $F_{j,i}$, and $G_{i,j}$ as these operators need the direct predecessor of some label, which is more or less distributed in step(p). Thus, we apply a Min_y operation to detect the direct predecessor of y in a step-language. As Min_y moves any y as far to the left as possible we have to undo this effect later by a $\vec{y}$ operation. We would like to get $\text{step}(G_y(L)) = \vec{y} \circ \text{step} \circ G_y \circ \text{Min}_y(\text{step}(L))$ – yet, this equation is invalid. G_y cannot identify the label of the interface transition as a y-label may have more than one direct predecessor in a step-word. Since the label of the interface transition is deleted anyway, we can restrict the G_y operation to step-words with only one direct predecessor step for each y-label. We define an operation G_y^* on step-words such that $\text{step}(G_y(L)) = \vec{y} \circ G_y^* \circ \text{Min}_y(\text{step}(L))$ holds by

$$G_y^*(\varepsilon) := \varepsilon, \qquad G_y^*(X_1) := X_1,$$

$$G_y^*(X_1X_2\chi) := \begin{cases} G_y^*(X_2\chi) & \text{if } |X_1| = 1 \text{ and } y \in X_2, \\ X_1 G_y^*(X_2\chi) & \text{if } y \notin X_2, \\ \perp & \text{otherwise.} \end{cases}$$

We define $\chi\perp := \perp =: \perp\chi$ for any step-word χ to ensure that step-words that can only be processed partly by G_y^* are removed from the step-language.

For $F_{j,i}$ we have the situation that an x_j^+ may become a predecessor of a label of the ith interface transition artificially when we apply the step-operation to a pomset. Analogous observation can be made for $G_{i,j}$. On the other hand, while with pomsets we have the problem that we may introduce forbidden circles in the ordering relation of a pomset, this problem simply does not exist with step-words. Therefore, we translate $F_{j,i}$ and $G_{i,j}$ directly to operations $F_{j,i}^*$ and $G_{i,j}^*$ on step-words such that

$$\text{step}(F_{j,i}(\text{step}(L))) = \vec{y}_i \circ F_{j,i}^* \circ \text{Min}_{y_i}(\text{step}(L))$$

and

$$\text{step}(G_{i,j}(\text{step}(L))) = \vec{y}_j \circ G_{i,j}^* \circ \text{Min}_{y_j}(\text{step}(L))$$

hold. To do so, we define $F^*_{j,i}$ as $\psi^{-1}_{j,i}$ with

$$\psi_{j,i}(\varepsilon) := \varepsilon, \qquad \psi_{j,i}(X_1) := X_1,$$

$$\psi_{j,i}(X_1X_2\chi) := \begin{cases} X_1\psi_{j,i}(X_2\chi) & \text{if } X_2(y_i) = 0, \\ (X_1 \cup \{k \cdot x_j^+\})\psi_{j,i}(X_2\chi) & \text{if } X_2(y_i) = k, \end{cases}$$

and $G^*_{i,j}$ as $\phi^{-1}_{i,j}$ with

$$\phi_{i,j}(\varepsilon) := \varepsilon, \qquad \phi_{i,j}(X_1) := X_1,$$

$$\phi_{i,j}(X_1X_2\chi) := \begin{cases} X_1\phi_{i,j}(X_2\chi) & \text{if } X_2(y_j) = 0, \\ (X_1 \cup \{k \cdot x_i^-\})\phi_{i,j}(X_2\chi) & \text{if } X_2(y_j) = k, \end{cases}$$

for non-empty steps X_1, X_2 and step-sequences χ.

For $M_{i,j;a}$ we even have to apply the Min operator twice. As step-languages are closed under unstepping we find some step-word of the form $\chi_1 X\{y_i, y_j\}\chi_2$ with $|X| = 2$. Assume $X = \{a', a''\}$. Thus, we have identified the two labels of the ith and jth interface transition to be a' and a'' (although we do not know whether a' is the label of the ith or the jth interface transition). To simulate a $meld^T_{i,j;a}$ we thus always have to replace pairs of a', a'' simultaneously by a new action name a in all steps preceding a $\{y_i, y_j\}$-step. We get

$$\text{step}(M_{i,j;a}(\text{step}(L))) = \vec{y}_i \circ M^*_{i,j;a} \circ \text{Min}_{y_i} \circ \text{Min}_{y_j}(\text{step}(L))$$

where $M^*_{i,j;a}$ is defined by

$$M^*_{i,j;a}(\varepsilon) := \varepsilon, \qquad M^*_{i,j;a}(X_1) := X_1,$$

$$M^*_{i,j;a}(X_1X_2\chi) := \begin{cases} X_1M^*_{i,j;a}(X_2\chi) & \text{if } X_2(y_i) = 0 = X_2(y_j), \\ (X_1 \cup \{k \cdot a\} - \{k \cdot a', k \cdot a''\})M^*_{i,j;a}((X_2 - \{k \cdot y_j\})\chi) & \\ & \text{if } \chi'\{a', a''\}\{y_i, y_j\}\chi'' \in \text{Min}_{y_i}\text{Min}_{y_j}(\text{step}(L)) \\ & \text{for some } \chi', \chi'', \text{ and } X_2(y_i) = k = X_2(y_j), \\ \perp & \text{otherwise}, \end{cases}$$

for non-empty steps X_1, X_2, and step-sequences χ.

We have presented translations for operations on sets of pomsets to operations on sets of step-words. Using these, we get the semantic counterparts $\widehat{\widehat{op}}_k$ in $\mathcal{S}^{step}$ of the syntactic operators op_k of $\mathcal{E}$ from $\widehat{op}_k$ of Theorem 1. For step-languages L, L_1, L_2:

$$\widehat{\widehat{0}} = \{\varepsilon\}, \qquad \widehat{\widehat{place}} = \bigsqcup\{\{x_1^+\}\{x_1^-\}, \{x_1^+\}\}, \qquad \widehat{\widehat{a\text{-}trans}} = \{y_1\} \sqcup \bigsqcup(\{a\}\{y_1\}),$$

$$L_1 \mathbin{\widehat{\widehat{\|}}} L_2 = L_1 \sqcup h_{L_1}(L_2), \qquad \widehat{\widehat{reorder}}_\pi(L) = \hat{\pi}(L),$$

$$\widehat{\widehat{hide}}^P_i(L) = h_{x_i}(L \cap STEP^*(Act - \{x_i^+, x_i^-\})), \qquad \widehat{\widehat{hide}}^T_i(L) = \bar{h}_{y_i}(L),$$

$$\widehat{\tau\text{-}hide}_i(L) = \bar{h}_{y_i} G^*_{y_i} \mathrm{Min}_{y_i}(L), \qquad \widehat{mark}_s(L) = \mathrm{delete}_s(L),$$

$$\widehat{arc}^{T\to P}_{i\to j}(L) = \vec{y}_i\, F^*_{j,i} \mathrm{Min}_{y_i}(L), \qquad \widehat{arc}^{P\to T}_{i\to j}(L) = \vec{y}_j\, G^*_{i,j} \mathrm{Min}_{y_j}(L),$$

$$\widehat{meld}^T_{i,j;a}(L) = \bar{h}_{y_j}\, \vec{y}_i\, M^*_{i,j;a} \mathrm{Min}_{y_i} \mathrm{Min}_{y_j}(L).$$

$\widehat{meld}^P_{i,j}(L)$ is again obtained directly from the syntactic construction described in Theorem 1. Further, $\mathrm{delete}_s(L) = \{w_s\} \backslash L$ holds, where w_s is the step $\{(x_1^+)^{s_1}, \ldots, (x_n^+)^{s_n}\}$ for $s = (s_1, \ldots, s_n)$. The reason is that any x_i^+ is always firable. Thus, deleting the "first s" elements or some arbitrary s-elements of x^+-transitions in L leads to the same set. □

Definition 19 (*Petri net language semantics*). $\mathscr{S} : \mathscr{PNI} \to \mathscr{L}_0$ is defined by

$$\mathscr{S}(N) := L_0(U[N]),$$

with

$$L_0(U[N]) := \{w \in Act^*;\ \exists \sigma \in F(U[N]) : \lambda_{U[N]}(\sigma) = w \text{ and}$$
$$\exists s : (s_{U[N]}[\sigma > s \text{ and no } y_j, 1 \leqslant j \leqslant |t_N| \text{ is firable in } s)\}.$$

$$\mathscr{L}_0 := \{L_0(U[N]) \mid N \in \mathscr{PNI}\}.$$

In contrast to the previous two semantics, we are not able to show compositionality of this interleaving semantics for $\mathscr{E}$ in general. Most semantic operations can easily be transferred from the step semantics, but it is impossible to present a semantic equivalent for the $meld^T$-operator. In Fig. 9 we present two Petri nets with the same interleaving semantics that behave differently after having applied a $meld^T_{1,2;c}$.

Therefore, we can conclude the following.

Corollary 1 (Non-compositionality of $\mathscr{S}$). *$\mathscr{S}$ is not compositional for $meld^T$.*

Corollary 2 (Compositionality of $\mathscr{S}$ except for $meld^T$). *$\mathscr{S}$ is compositional for $\mathscr{E}$ with the exception of $meld^T$.*

Obviously, $\mathscr{L}_0 = \mathrm{lin}(\mathscr{L}_0^{step})$ and $\mathscr{S} = \mathrm{lin} \circ \mathscr{S}^{step} = \mathscr{S}^{step} \cap Act^*$ hold. Furthermore, for all operators $op^k : (\mathscr{L}_0^{step})^k \to \mathscr{L}_0^{step}$ except $M^*_{i,j;a}$ and Min_y (as used in the proof of Theorem 2) it is true that

$$\mathrm{lin} \circ op^k(\mathrm{lin}(L_1), \ldots, \mathrm{lin}(L_k)) = \mathrm{lin}(op^k(L_1, \ldots, L_k)) \tag{3}$$

holds for $L_1, \ldots, L_k \in \mathscr{L}_0^{step}$. We replace each op^k by $\overline{op}^k : (\mathscr{L}_0)^k \to \mathscr{L}_0$ with $\overline{op}^k := op^k \cap Act^*$. The Min_y-operator cannot be transferred from step-words to words as is easily seen: Suppose $\{a, b\}\{y\}$ is minimal with respect to y and let y be dependent on a only. Due to the concurrency of a and b the expression $\mathrm{lin}(\{a, b\}\{y\})$ not only

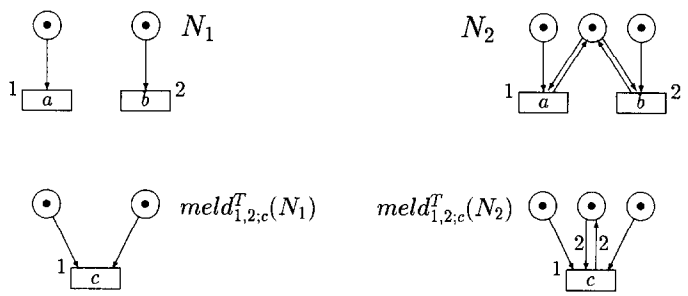

Fig. 9. Two Petri nets N_1, N_2 with the same interleaving semantics $\mathcal{S}(N_1) = \mathcal{S}(N_2) = \{\varepsilon, ay_1\} \sqcup\!\sqcup \{\varepsilon, by_2\}$. After a $meld^T_{1,2;c}$ operation the semantics differ: $\mathcal{S}(meld^T_{1,2;c}(N_1)) = \{\varepsilon, cy_1\} \neq \{\varepsilon\} = \mathcal{S}(meld^T_{1,2;c}(N_2))$.

yields the word *bay* but also *aby* which is not minimal with respect to y anymore. We avoid this by directly using the Min_y-operator for words, i.e. we replace the operator Min_y by $\mathrm{Min}_y \circ \mathrm{lin}$. Overall, $\mathcal{S}$ is a special case of $\mathcal{S}^{step}$ except for $M^*_{i,j;a}$ and thus compositional for $\mathcal{E}$ without the $meld^T$ operation.

6. Applications

One question which naturally arises concerning the expressiveness of this calculus is: How does it relate to other calculi of nets? We will especially consider the Box calculus [13] and Milner's Action calculus [15].

First, we take a short look at the Box calculus. The semantics of a program in a simple, parallel programming language $B(PN)^2$ is described by a combination of so-called Petri Boxes. A Petri Box is a safe Petri net with distinguished sets of input and output places. Usually, a Box begins firing with a token on each of its input places and terminates with one token on each output place. Petri Boxes simulate the control flow of a program as well as the data the program operates on. They can be combined by operators modelling the typical constructs in parallel programming like sequential and parallel composition, non-deterministic choice, loops, synchronization, and so on. Elementary Petri Boxes, from which other Petri Boxes are derived, are Boxes consisting of one input place, one output place and (in general) one transition describing the action that has to take place. Variables are simulated by special Data Boxes; assignments are handled by synchronization between transitions of the control flow and transitions of a Data Box. We show in the following how the operations of sequential composition and non-deterministic choice can be handled in our calculus $\mathcal{E}$ by the use of contexts. As we already have two sets of distinguished elements in a Petri net that can act as input and output we make a minor change to the concept of Boxes: instead of output places we consider output transitions.

As a first example we present the sequential composition. Sequential composition of two nets has been defined in [13] and in [4] where a certain output has to be produced

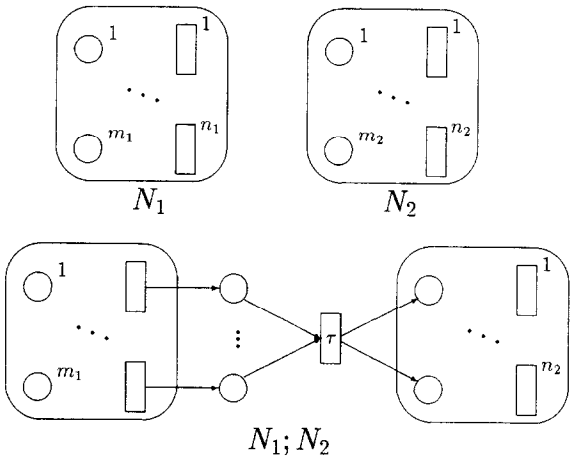

Fig. 10. Two arbitrary Petri nets N_1 and N_2 with $\dim(N_1)=(m_1, n_1)$, $\dim(N_2)=(m_2, n_2)$ and their sequential composition $N_1; N_2$.

in the first net before the second net can begin firing. In a simpler form, this operation also appears in the SCONE calculus [8] as the prefixing of an expression by an action. We describe the sequential composition as follows: Let N_1 and N_2 be nets with dimensions (m_1,n_1) and (m_2,n_2), respectively. The sequential composition $N_1;N_2$ has dimension (m_1,n_2); an interface place of N_2 receives one token if each of the interface transitions of N_1 has fired once. This idea is graphically presented in Fig. 10. Obviously, the sequential composition operator ";" depends on the dimensions of the nets N_1 and N_2. We define a parametrized syntactic operator "$;^{m_1,n_1}_{m_2}$", which is applicable to two Petri nets of dimension (m_1,n_1) and (m_2,n_2) for arbitrary n_2, by

$$N ;^{m_1,n_1}_{m_2} N' := (P \dot\cup P' \dot\cup \{p_1,\ldots,p_{n_1}\}, T \dot\cup T' \dot\cup \{t\}, F \cup F' \cup F_{seq},$$
$$\lambda \cup \lambda' \cup \{t \mapsto \tau\}, s \cup s', \boldsymbol{p}, \boldsymbol{t}'),$$

where $F_{seq} := \{(\boldsymbol{t}(k), p_k) \mid 1 \leqslant k \leqslant n_1\} \cup \{(p_k, t) \mid 1 \leqslant k \leqslant n_1\} \cup \{(t, \boldsymbol{p}'(k)) \mid 1 \leqslant k \leqslant m_2\}$. We can express these operators in $\mathscr{E}$ by an appropriate context (with $\|$, therefore we need the parameter m_1) by (compare Fig. 10):

$$SEQ^{m_1,n_1}_{m_2} := hide^T_{\{1,\ldots,n_1\}} hide^P_{\{1,\ldots,n_1\}} arc^{T\to P}_{(1,\ldots,n_1)\to(1,\ldots,n_1)} \tau\text{-}hide_1 hide^P_{\{m_1+n_1+1,\ldots,m_1+n_1+m_2\}}$$
$$arc^{T\to P}_{(1,\ldots,1)\to(m_1+n_1+1,\ldots,m_1+n_1+m_2)} (arc^{P\to T}_{(1,\ldots,n_1)\to(1,\ldots,1)} (place^{n_1} \| a\text{-}trans) \| \bullet).$$

Obviously, $N_1;N_2 = N_1 ;^{m_1,n_1}_{m_2} N_2 = SEQ^{m_1,n_1}_{m_2}[N_1 \| N_2]$, for nets N_i of dimension (m_i,n_i). As we have found an expression in $\mathscr{E}^\bullet$ for each operator $;^{m_1,n_1}_{m_2}$ these operators are compositional in our semantics. Additionally, we have seen that the dimension of a Petri net can be determined from its semantics. This means, given only the semantics $\mathscr{S}(N)$

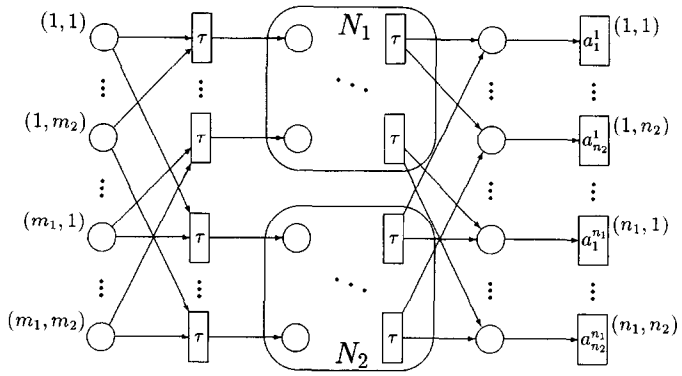

Fig. 11. The choice $N_1 \square N_2$. Input and output are formed as a direct product of the inputs and outputs of N_1 and N_2. An input place marked with (i,j) is the $((i-1)\cdot m_2 + j)$th place of the interface, an output transition (i,j) gets position $((i-1)\cdot n_2 + j)$ in the interface.

and $\mathcal{S}(N')$ of two arbitrary nets N and N' we can decide which operator $;_{m_2}^{m_1,n_1}$ to apply for the sequential composition $N;N'$. Thus, we can determine the semantics of $N;N'$. Therefore, also the non-parametrized sequential composition ; becomes compositional.

The choice operator $\square$ decides non-deterministically with which of two given nets N_1 and N_2 to continue. The choice is made at the moment of the first firing of a transition of either N_1 or N_2. Once a transition of N_1 has fired, N_2 must be disabled, and vice versa. One can easily express this by using a product of the input places of N_1 and N_2. This is depicted in Fig. 11. Firing the first τ-transition leading into N_1, e.g., consumes the tokens on the places $(1,1)$ to $(1,m_2)$, disabling all transitions that could produce tokens on the interface places of N_2. The output works in a similar way; if each (former) interface transition of, say, N_1 has fired once, each of the interface transitions $(1,1)$ to (n_1,n_2) may also fire once. A context $CHOICE_{m_2,n_2}^{m_1,n_1}$ for this construction can obviously be derived. In analogy to the sequential composition we conclude $CHOICE_{m_2,n_2}^{m_1,n_1}[N_1 \,\|\, N_2] = N_1 \square N_2$ and get compositionality even for the non-parametrized operator $\square$.

As a further example we want to take a look at the action calculus for Petri nets of Milner [15]. Milner constructs Petri nets from operations $pre_{\boldsymbol{p}}^t$, $post_{\boldsymbol{p}}^t$, and $mark_{\boldsymbol{p}}$ for transition names t and vectors of places $\boldsymbol{p}$. A local binding operation vt declares the pre- and postset of a transition t to be fully defined. The first three operations may be combined by parallel product while the local binding applies to the whole expression derived so far. A distributive law allows for the rearrangement of operators: $vt(E \,\|\, mark_{\boldsymbol{p}}) = vt(E) \,\|\, mark_{\boldsymbol{p}}$, $vt(E \,\|\, pre_{\boldsymbol{p}}^{t'}) = vt(E) \,\|\, pre_{\boldsymbol{p}}^{t'}$, and an analogous equation for $post_{\boldsymbol{p}}^{t'}$, for an expression E and transitions t, t' with $t \neq t'$. We may write, e.g., the Petri net N_2 of Fig. 3 as $va(pre_{p_1,p_3}^a \| post_{p_3,p_4}^a) \,\|\, vb(pre_{p_2,p_3}^b \| post_{p_3,p_5}^b) \,\|\, mark_{p_1,p_2,p_3}$ in Milners calculus. The operations $pre_{\boldsymbol{p}}^t$, $post_{\boldsymbol{p}}^t$, and $mark_{\boldsymbol{p}}$ can be expressed by our operations $arc^{P\to T}$, $arc^{T\to P}$, and $mark$, e.g., the expressions $pre_{p_1,p_2}^t \| post_{p_3}^t \| mark_{p_1,p_2}$

and $arc^{P\to T}_{(1,2)\to(1,1)} arc^{T\to P}_{1\to 3}\, mark_1 mark_2(place^3 \,\|\, t\text{-}trans)$ represent the same Petri net. Clearly, we must provide a *place* or *t-trans* operation in our calculus whenever a formerly unused name p or t shows up. After a local binding, no further arcs can be added to a transition, so vt is equivalent to a $hide^T$-operator for the transition t. We get different effects from Milners parallel operator depending on whether a local binding has already been applied to a transition t or not. While $vt(pre^t_p) \,\|\, vt(post^t_{p'})$ represents a net with two transitions, $vt(pre^t_p \,\|\, post^t_{p'})$ has only one transition. In the former expression there are two (possibly concurrent) transitions, both with name t, as we would expect from a parallel operator. In the latter formula, though, the two transitions are merged into one, just as if we had applied a $meld^T$ operation. Milners parallel product consists of a mixture of synchronization and disjoint union, which we denote by the symbol $\|_{sy}$. Without loss of generality we assume that unbound transitions (those outside the scope of a local binding) in an expression have names $t_1, t_2, \ldots$, and so forth. We can then interpret t_1 to be the first interface transition, t_2 the second, and so on. It is rather simple to explain the parallel product $\|_{sy}$ using a context in $\mathscr{E}$. For the nets N_1 and N_2 of Fig. 10 let $n := \min\{n_1, n_2\}$ and define $SYNC^{n_1}_{n_2} := meld^T_{1,n_1+1;t_1} \ldots meld^T_{n,n_1+n;t_n}(\bullet)$. It is easily seen that $SYNC^{n_1}_{n_2}[N_1 \,\|\, N_2] = N_1 \|_{sy} N_2$. Again, as we can detect the required parameters in the semantics, even the non-parametrized $\|_{sy}$-operator of Milner becomes compositional in our semantics.

Operations of other calculi allowing for the synchronization of two transitions can be modelled in the same way. Such operations exist, e.g., in the calculi of Best [13] or Gorrieri [8].

In general, many operations from calculi for Petri nets can be expressed using the same technique: Construct the disjoint union of the participating Petri nets by a $\|$-operator and embed this construction into an appropriate context. There is a common form of operation that cannot be derived in this way: a recursion operator. Such an operator generally creates an infinite net from a given finite one.

We conclude this chapter with a remark on removing parts of a Petri net; this is only supported by the operations $meld^T$ and $meld^P$ in our calculus. Other operations for removing structure cannot be performed directly. We are mainly interested in the semantics of a Petri net and not in its structure. For example, if we want to remove a transition from a Petri net we really would like to have a behaviour in which the transition cannot fire. This can be done easily; we introduce a new place into the net, attach it to our transition by an $arc^{P\to T}$-operator, and hide the place. The transition becomes dead and cannot be detected in any way. Overall, we get the same effect in the behaviour that removing the transition would have yielded. In a similar way, we can try to handle the removal of a place. Instead of eliminating a place from a net, which cannot be done in our calculus, we introduce a new transition, attach it to the place using an $arc^{T\to P}$-operator, and τ-*hide* it. The production of tokens on the place by the new τ-transition is unobservable, so we have a potentially infinite number of tokens available. No transition is disabled because of too few tokens on this place. Thus, removal of a place or feeding arbitrary many tokens onto this place are semantically equivalent for many semantics. However, this does not work for the pomset behaviour;

in a pomset we may be able to distinguish between tokens of different sources by a look at the causality relation.

7. Maximal behaviours

There is some serious objection to a language semantics for Petri nets due to the fact that for any Petri net there is a Free Choice net (see, e.g., [11]) with the same language. A place, p, is called *shared* if there are different transitions t_1, t_2 with $F(p,t_1)>0$ and $F(p,t_2)>0$, i.e., p is an input place for several transitions. A Petri net is called a Free Choice net if only those transitions with exactly one input place may possess a shared input place. In other words, if some transition t possesses several input places none of them is shared. As a consequence, any token in a Free Choice net on a shared input place of transitions t_1, t_2 possesses the free choice to decide whether "it fires t_1 or t_2". Free Choice nets have to guess how to solve conflicts while Petri nets may control conflicts. However, a wrong guess in a Free Choice net can be hidden in a deadlock, i.e. a prefix of the original firing-sequence, and is therefore invisible in prefix-closed languages. The same objection holds for step-languages and pomset-languages. Fig. 12 shows the idea of this technique of simulating an arbitrary Petri net (pomset-, step-) language by a Free Choice net (pomset-, step-) language. This strange equivalence is avoided by *maximal* behaviours.

Definition 20 (*Maximal behaviours*). Let $\surd$ be a special symbol in Act, not used before. By $p\surd$ we denote the word $p\surd$ in case of $p \in L(N)$, the step-word $p\{\surd\}$ in case of $p \in L^{step}(N)$, and the pomset $[(X_p \cup \{\surd\}, <_p \cup \{x < \surd \mid x \in X_p\}, \ell_p \cup \{\surd \mapsto \surd\})]$ in case of $p \in L^{pom}(N)$. The maximal behaviours of N are defined by

$$L^{(step,)max}(N) := L^{(step)}(N) \cup \{\lambda_N(\sigma)\surd \mid \sigma \in F^{(step)}(N) \text{ and } \sigma \text{ is maximal}\},$$

$$L^{pom,max}(N) := L^{pom}(N) \cup \{\Phi(\pi)\surd \mid \pi \text{ is a maximal process of } N\}.$$

N_1 N_2

$$\begin{aligned} \mathcal{S}(N_1) &= \{\epsilon, a\} &&= \mathcal{S}(N_2) \\ \mathcal{S}^{step}(N_1) &= \{\{\epsilon\}, \{a\}\} &&= \mathcal{S}^{step}(N_2) \\ \mathcal{S}^{pom}(N_1) &= \{(\epsilon), (a)\} &&= \mathcal{S}^{pom}(N_2) \end{aligned}$$

Fig. 12. A Petri net N_1 with an equivalent Free Choice net N_2 where N_2 "solves" conflicts by guessing and hiding a wrong guess in an invisible deadlock.

$\mathscr{L}^{(step,pom,)max} := \{L^{(step,pom,)max}(N) \mid N \in \mathscr{PNI}\}$ is the class of maximal (step-, pomset-) languages of Petri nets.

Unfortunately, we cannot give a compositional semantics for our calculus $\mathscr{E}$ for maximal behaviours. Consider an arbitrary net expression N and the constructions $N' := (place \,\|\, t\text{-}trans) \,\|\, N$ and $N'' := hide_1^P arc_{1\to 1}^{P\to T}(N')$. As the newly introduced transition t of N' is disabled in N'', N and N'' show the same behaviour. In N' though, the transition t is always firable, meaning the behaviour of N' does not contain any maximal elements (with a $\surd$). Therefore, we have to be able to determine all maximal elements of the behaviour of N'' directly from the non-maximal behaviour of N''. This cannot be done for general Petri nets but for a subclass of so-called *basic* nets.

Definition 21 (*Basic nets*). A Petri net $N = (P, T, F, \lambda, s_0, \boldsymbol{p}, \boldsymbol{t})$ is called *basic* if N possesses no arcs from transitions to places and all of its places and transitions are public, i.e., belong to the interface.

A basic net can be constructed solely by use of the operators *place*, *a-trans*, $\|$, and $arc_{i\to j}^{P\to T}$ of the calculus $\mathscr{E}$. We split our calculus $\mathscr{E}$ into two levels, the first for the construction of basic nets and the second level with the remaining operators, such that we can still build all possible Petri nets.

Definition 22 (*A calculus $\mathscr{E}^{max}$ for maximal behaviours*). $\mathscr{E}^{max}$ is a two-level calculus over Act defined by
Level 1: $\forall a \in Act : \forall i, j \in \mathbb{N}$:

$$BN :\equiv 0 \mid place \mid a\text{-}trans \mid BN \| BN \mid arc_{i\to j}^{P\to T}(BN)$$

Level 2: $\forall a \in Act : \forall i, j \in \mathbb{N} : \forall s \in \mu^{\mathbb{N}} : \forall \pi \in \gamma \times \gamma$:

$$\begin{aligned} PN :\equiv\ & BN \mid PN \,\|\, PN \mid arc_{i\to j}^{T\to P}(PN) \mid mark_s(PN) \mid reorder_\pi(PN) \\ & \mid hide_i^P(PN) \mid hide_i^T(PN) \mid \tau\text{-}hide_i(PN) \mid meld_{i,j;a}^T(PN) \end{aligned}$$

This new calculus drastically restricts the use of contexts. Normally, nets and contexts are expressed with operations of both levels of the calculus. This of course means that a Petri net can be embedded into a context only by using operations of level 2 now, explicitly forbidding the use of $arc^{P\to T}$-operators. A Petri net can therefore only be connected to a context by arcs from transitions to places and not by arcs from places to transitions. Such contexts are discussed in some detail in [16]. Also, some modular decomposition techniques use this concept of connecting parts of Petri nets, see, e.g., [20].

We cannot express the $meld_{i,j}^P$-operator as before by introducing two new τ-actions with the ith and jth interface place as pre- and post-places. Also, we cannot use x^--transitions as before in the universal context.

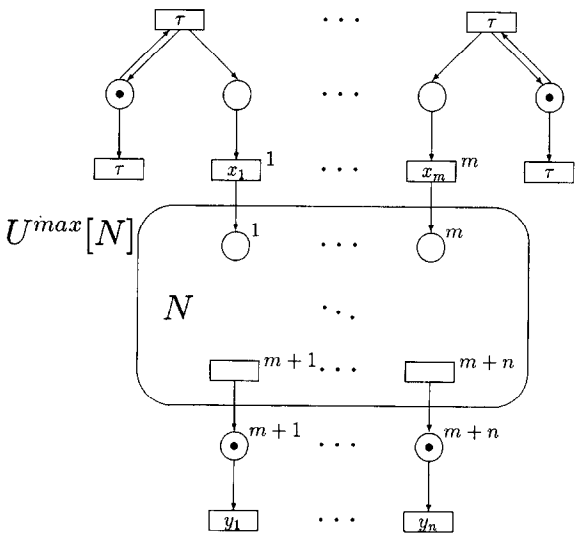

Fig. 13. Some net N with dimension (m,n) embedded into its universal context $U^{max}_{m,n}$.

Additionally, in maximal behaviours the x_i^+-transitions (we will write just x_i for x_i^+ from now on) of U have to fire infinitely. However, we must still be able to simulate those contexts that send only a finite number of tokens to N. Therefore, we add τ-transitions to U that can disable the x_i-transitions at any point of the simulation. The universal context suggested in [16] works well for interleaving semantics, but as we also investigate pomset semantics with autoconcurrency, our universal context must be defined as follows.

Definition 23 (*Universal context for maximal behaviours*). The universal context for a maximal behaviour of dimension (m,n) (as presented in Fig. 13) is defined by

$$U^{max}_{m,n} := arc^{T\to P}_{\boldsymbol{m+n}\to \boldsymbol{m+n}}(U^{\mathscr{X},max}_{m,n} \parallel \bullet \parallel U^{\mathscr{Y}}_{m,n})$$

with

$$U^{\mathscr{X},max}_{m,n} := hide^P_{\boldsymbol{m}}\tau\text{-}hide_{\boldsymbol{c}_m} arc^{T\to P}_{\boldsymbol{m}\to \boldsymbol{m}} hide^P_{\boldsymbol{m}} mark_{\{1,\ldots,m\}} arc^{T\to P}_{\boldsymbol{m}\to \boldsymbol{m}}\tau\text{-}hide_{\boldsymbol{d}_m}$$
$$arc^{P\to T}_{\boldsymbol{m}\to \boldsymbol{m}}(\boldsymbol{d}_m\text{-}trans \parallel arc^{P\to T}_{\boldsymbol{2m}\to \boldsymbol{2m}}(place^{2m} \parallel \boldsymbol{c}_m\text{-}trans \parallel \boldsymbol{x}_m\text{-}trans))$$

and $U^{\mathscr{Y}}_{m,n}$ from Definition 16. Again, we denote $U^{max}_{m,n}[N]$ by $U^{max}[N]$ for a Petri net N of dimension (m,n).

We again use an $L_0^{(step,pom)}$-behaviour for our semantics where no y_j-transitions of the universal context may remain enabled. This condition is trivially fulfilled for all maximal elements of the behaviour.

Definition 24 (*Maximal semantics*). The maximal semantics $\mathcal{S}^{(step,pom),max}$ are defined for a Petri net $N \in \mathcal{PNI}$ by $\mathcal{S}^{(step,pom),max}(N) := L_0^{(step,pom),max}(U^{max}[N]) := L_0^{(step,pom)}(U^{max}[N]) \cup (L^{(step,pom),max}(U^{max}[N]) \cap (POM^*\surd))$.

We get the following compositionality result.

Theorem 3 (Compositionality of maximal semantics). *The semantics* $\mathcal{S}^{pom,max}$ *and* $\mathcal{S}^{step,max}$ *are compositional for* $\mathcal{E}^{max}$, *and* $\mathcal{S}^{max}$ *is compositional for* $\mathcal{E}^{max}$ *without* $meld^T$.

Proof. The proofs of the compositionality of $\mathcal{S}$, $\mathcal{S}^{step}$, and $\mathcal{S}^{pom}$ are modified for $\mathcal{S}^{max}$, $\mathcal{S}^{step,max}$, and $\mathcal{S}^{pom,max}$ as follows. The semantics $\widehat{a\text{-}trans}$ of *a-trans* remains the same: $y_1 \sqcup \bigsqcup(a \to y_1)$, $\text{step}(y_1 \sqcup \bigsqcup(a \to y_1))$, and $y_1 \sqcup \bigsqcup a y_1$, respectively, as there are no finite maximal behaviours for *a-trans*. $\widehat{place}$ has to be changed in multiple ways, as not only the x_1^--transition is missing in the universal context but also x_1 may become dead after any number of firings due to an invisible withdrawal of the token in the $U^{\mathcal{X},max}$-part. Thus $\widehat{place}$ becomes $\bigsqcup x_1 \cup (\bigsqcup x_1)\surd$, $\text{step}(\bigsqcup x_1) \cup (\text{step}(\bigsqcup x_1))\surd$, and $x_1^* \cup x_1^*\surd$, respectively. Notice that the operations $reorder_\pi$, $hide^T$, and τ-*hide* do not influence the token game of a Petri net. Then, of course, they have no influence on maximality either. This also holds for $meld^T_{i,j;a}$. We have to remember here that pomsets with an unbalanced number of labels y_i and y_j are removed anyway. For pomsets where this number is balanced, replacing the ith and jth interface transition by a new one does not change the production and consumption of tokens. An operation $mark_{1\cdot p(i)}$ adds a token to the ith interface place. The corresponding $\widehat{mark}_{1\cdot p(i)}$ operation simulates this by removing an x_i-node from each pomset – leaving the token game and maximality unchanged. An analogous situation holds for the $arc^{T\to P}$ operation. A $hide_i^P$ removes the part of the universal context that is attached to the ith interface place. Let τ_1 denote the τ-transition that produces tokens for the x_i-transition and τ_2 the transition that disables τ_1. Each pomset represents a behaviour with enabled as well as one with disabled τ_1-transition, see $\widehat{place}$. Suppose, removing this part of the universal context makes a behaviour maximal (which happens if the $\tau_1/\tau_2/x_i$-transitions were the only transitions firable). Then this behaviour has already been maximal (with the right number of firings of τ_1 and τ_2 (once)). Therefore, the behaviour with a $\surd$ already exists in the semantics. Thus, the semantic operations $\widehat{\cdot}^{max}$ in the maximal semantics are trivially obtained from their counterparts $\widehat{\cdot}$ in the non-maximal semantics for the syntactic operators *reorder*, $hide^P$, $hide^T$, τ-*hide*, *mark*, and $arc^{T\to P}$:

$$\widehat{\cdot}^{max}(p\surd) := (\widehat{\cdot}(p))\surd,$$
$$\widehat{\cdot}^{max}(p) := \widehat{\cdot}(p), \quad \text{for } p \in POM^* \text{ (or } STEP^* \text{ or } Act^*, \text{ respectively}).$$

For $\|$ we obviously use

$$\widehat{\|}^{max}(p_1\surd, p_2\surd) := (\widehat{\|}(p_1, p_2))\surd,$$
$$\widehat{\|}^{max}(p_1\surd, p_2) = \widehat{\|}^{max}(p_1, p_2\surd) = \widehat{\|}^{max}(p_1, p_2) := \widehat{\|}(p_1, p_2),$$

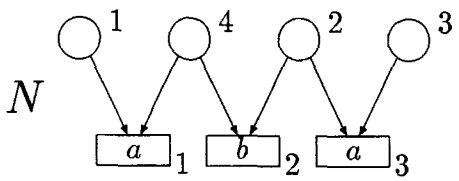

Fig. 14. Example of a net N in BN.

for $p_1, p_2 \in POM^*$ (or $STEP^*$ or Act^*, respectively), as for a maximal parallel behaviour both involved behaviours have to be maximal.

However, the $arc_{i\to j}^{P\to T}$-operator leads to severe difficulties in maximal semantics. The reason is that a new arc from place p_i to transition t_j may block this transition from firing. This may lead to a "local maximality". However, whether the new overall behaviour is maximal now depends on further possible activities outside from t_j. Thus, when we apply the semantic counterpart $\widehat{arc}_{i\to j}^{P\to T,max}$ to a pomset p (or step-word χ, or word w) without a $\surd$ at the end we may receive a pomset p' (or χ', or w') where the decision whether p' ends with a $\surd$ may depend on global properties of the underlying net N. However, N is unknown in the semantic world. To overcome this inherent difficulty we have designed the two levels of $\mathscr{E}^{max}$ in such a way that all basic nets of level 1 (where we may apply an $arc_{i\to j}^{P\to T}$-operator solely) have a nice "testing-for-maximality"-property. As stated before, for any basic net $N \in BN$ and any $p \in \mathscr{S}^{pom}(N)$, $\chi \in \mathscr{S}^{step}(N)$, and $w \in \mathscr{S}(N)$ we do not need to know N in order to test whether p, χ, and w may represent maximal behaviours. Fig. 14 presents an example. Obviously, $\mathscr{S}^{pom}(N) = \coprod(x_1 \sqcup x_2 \sqcup x_3 \sqcup x_4) \sqcup \coprod(\begin{smallmatrix}x_1\\x_4\end{smallmatrix} \searrow a) \sqcup \coprod(\begin{smallmatrix}x_4\\x_2\end{smallmatrix} \searrow b) \sqcup \coprod(\begin{smallmatrix}x_2\\x_3\end{smallmatrix} \searrow a)$.

Any $p \in \mathscr{S}^{pom}(N)$ is the abstraction of a maximal process if there remains no pair of nodes (x_2,x_3), (x_1,x_4), or (x_2,x_4) without a successor node. Thus, for any "maximal" $p \in \mathscr{S}^{pom}(N)$ we simply add $p\surd$ to the semantics. This gives us a general principle on how to handle a compositional maximal behaviour on level 1: Use the compositional behaviour without maximality (i.e., $\mathscr{S}^{pom}$, $\mathscr{S}^{step}$, or $\mathscr{S}$, respectively) and add $p\surd$ ($\chi\surd$, $w\surd$) to the behaviour for those p (χ, w) that may represent maximal behaviours. We omit the rather trivial details. □

Of course, the Petri nets N_1 and N_2 of Fig. 12 differ in their maximal behaviours as $\varepsilon\surd \notin \mathscr{S}^{(step,\,pom,)max}(N_1)$ but $\varepsilon\surd \in \mathscr{S}^{(step,\,pom,)max}(N_2)$ due to the possible wrong guess of N_2.

8. Summary

We have enriched Petri nets with an interface and with an algebraic structure $\mathscr{E}$. In this framework, a Petri net context becomes an expression in $\mathscr{E}$ with an additional variable. We have investigated a true-concurrency pomset behaviour for Petri

nets and, as special cases, an interleaving behaviour and some intermediate step-behaviour where concurrently enabled transitions may fire simultaneously. We have found a universal context U for each of those behaviours B_i and could prove the semantics $\mathcal{S}_i := B_i(U[N])$ to be compositional. Additionally, we have presented a modified two-level calculus $\mathcal{E}^{max}$ that yields compositional semantics for the corresponding maximal behaviours B_i^{max}. Thus, while $\mathcal{S}_i'(N) := B_i(N)$ presents only some naive semantics without interesting algebraic properties we suggest the definition of a semantics as the behaviour of a net embedded in its universal context. This slight change results in compositionality.

Originally, we first found the compositional semantics for interleaving languages, "extended" this result to step-languages, and, finally, to pomset-languages, noticing that the proofs get easier for the more general pomset case. We could also present a calculus $\mathcal{E}'$, similar to $\mathcal{E}$, for free Petri nets with interfaces, such that $\mathcal{S}$ also proves to be compositional for $\mathcal{E}'$. This has led to a new proof of Hack's algebraic Petri net languages characterization, see [11], and to an algebraic characterization for $\mathcal{L}^{pom}$, [23].

Acknowledgements

We would like to thank Eike Best, Andrea Corradini, Pierpaolo Degano, Katrin Erk, Javier Esparza, Ugo Montanari, Mogens Nielsen, Vladimiro Sassone and Steve Schneider for valuable discussions that have led to these results. Also, the comments of an unknown referee have been very helpful. In fact, they have led to a complete rearrangement of the paper.

References

[1] E. Best, C. Fernandéz, Nonsequential processes: a Petri net view, Monographs on Theoretical Computer Science, vol. 13, Springer, Berlin, 1988.

[2] E. Best, R. Pinder Hopkins, $B(PN)^2$ – a basic Petri net programming notation, Lecture Notes in Computer Science, vol. 694, Springer, Berlin, 1993, pp. 379–390.

[3] C. Brown, D. Gurr, V. de Paiva, A linear specification language for petri nets, Report DAIMI PB-363, Computer Science Dept., University of Aarhus, 1991.

[4] L. Cherkasova, Posets with non-actions: a model for concurrent nondeterministic processes, Arbeitspapiere der GMD 403, Gesellschaft für Mathematik und Datenverarbeitung mbH, 1989.

[5] J.W. de Bakker, J.H.A. Warmerdam, Metric pomset semantics for a concurrent language with recursion, Lecture Notes in Computer Science, vol. 469, Springer, Berlin, 1990.

[6] P. Degano, R. de Nicola, U. Montanari, A distributed operational semantics for CCS based on condition/event systems, Acta Inform. 26 (1988) 59–91.

[7] U. Goltz, On representing CCS programs by finite Petri nets, Lecture Notes in Computer Science, vol. 324, Springer, Berlin, 1988, pp. 339–350.

[8] R. Gorrieri, U. Montanari, SCONE: a simple calculus of nets, Lecture Notes in Computer Science, vol. 458, Springer, Berlin, 1990, pp. 2–30.

[9] J. Grabowski, On partial languages, Fund. Inform. IV.2 (1981) 427–498.

[10] S.A. Greibach, Remarks on blind and partially blind one-way multicounter machines, Theoret. Comput. Sci. 7 (1978) 311–324.

[11] M. Hack, Petri net languages, Computation Structures Group Memo 124, Project MAC, MIT, 1975.

[12] M. Jantzen, Language theory of Petri nets, Lecture Notes in Computer Science, vol. 254, Springer, Berlin, 1987, pp. 397–412.
[13] M. Koutny, J. Esparza, E. Best, Operational semantics for the Petri Box calculus, Lecture Notes in Computer Science, vol. 694, Springer, Berlin, 1994, pp. 210–225.
[14] A. Mazurkiewicz, Trace theory, Lecture Notes in Computer Science, vol. 255, Springer, Berlin, 1987, pp. 279–324.
[15] R. Milner, Action calculi, or syntactic action structures, Lecture Notes in Computer Science, vol. 711, Springer, Berlin, 1993, pp. 105–121.
[16] M. Nielsen, L. Priese, V. Sassone, Characterizing behavioural congruences for Petri nets, in: I. Lee, S. Smolka (Eds.), Lecture Notes in Computer Science, vol. 962, Proc. CONCUR '95, Springer, Berlin, 1995, pp. 175–189.
[17] E.R. Olderog, Operational Petri net semantics for CCSP, Lecture Notes in Computer Science, vol. 266, Springer, Berlin, 1987, pp. 196–223.
[18] J.L. Peterson, Petri nets, Comput. Surveys 9 (3) (1977) 223–252.
[19] L. Pomello, G. Rozenberg, C. Simone, A survey of equivalence notions for net based systems, Lecture Notes in Computer Science, vol. 609, Springer, Berlin, 1992, pp. 410–472.
[20] L. Priese, Automata and concurrency, Theoret. Comput. Sci. 25 (1983) 221–265.
[21] P.H. Starke, Traces and semiwords, Newsletter No. 19, Bonn, Germany: Gesellschaft für Informatik (GI), Special Interest Group on Petri Nets and Related System Models, 1985.
[22] D. Taubner, Theoretical CSP and formal languages, Report TUM-I8706, Institut für Informatik, Technische Universität München, 1987.
[23] H. Wimmel, L. Priese, Algebraic characterization of Petri net pomset semantics, Proc. CONCUR'97, Lecture Notes in Computer Science, vol. 1243, Springer, Berlin, 1997, pp. 406–420.